UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

TECNOLOGÍA ESPECÍFICA DE COMPUTACIÓN

TRABAJO FIN DE GRADO

Plataforma para la Edición Líquida Multiformato

María Álvarez Rodríguez

Septiembre, 2016

PLATAFORMA PARA LA EDICIÓN LÍQUIDA MULTIFORMATO

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

DEPARTAMENTO DE TECNOLOGÍAS Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

Plataforma para la Edición Líquida Multiformato

Autor: María Álvarez Rodríguez

Director: Dr. Carlos González Morcillo

Septiembre, 2016

Ciudad Real – Spain

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

Resumen

"La información es poder", este pensamiento, atribuido a Francis Bacon, simboliza la evolución que ha sufrido el hombre para llegar a su posición actual. El hombre es un ser ávido de conocimiento, y esto es un hecho que ha quedado patente desde las primeras representaciones pictóricas, donde el ser humano sentía la necesidad de transmitir sus vivencias. Esta labor es cada vez es más fácil, como consecuencia de un proceso acelerado de evolución en los últimos años, con el desarrollo y perfeccionamiento de nuevas tecnologías y herramientas, que han propiciado la aparición de instrumentos como libros digitales o tablets, pero sobre todo, con Internet.

Es por ello, que los sistemas de publicación deben adaptarse a estas nuevas características, para abarcar a un público lo más amplio posible. Esta situación conlleva un sobreesfuerzo por parte del editor que debe ocuparse de publicar el material en distintos formatos, pudiendo ser completamente distintos y requerir desarrollos muy dispares.

HIDRA surge de esta necesidad y su principal objetivo está enfocado en facilitar esta tarea, para minimizar esfuerzos y conseguir resultados más eficientes. Es por ello que el sistema ofrece soporte a la gestión de proyectos y proporciona un conjunto de estructuras típicas de maquetación disponibles para aplicar en los mismos.

Por otro lado, HIDRA ofrece la posibilidad de procesar un proyecto para obtener diferentes formatos de salida distintos dependiendo de la necesidad del usuario. Además, tiene definidos una serie de estilos y plantillas que pueden ser modificados para adaptare a cualquier necesidad de generación futura.

Abstract

'Information is power', this thought, attributed to Francis Bacon, symbolizes the evolution that the man has suffered to reach his current position. The man is an avid knowledge being, and this is a fact that has become clear from the first pictorial representations, where the man felt the need to transmit their experiences. And it is becoming easier as a result of an accelerated in recent years, with the development and refinement of new technologies and tools, which have promoted the emergence of instruments like digital books or tablets, but above all, with the Internet.

It is for this reason that publishing systems must adapt to these new features, to cover a public as broad as possible. This situation involves an extra effort by the publisher to be addressed by publishing the material in different formats and can be completely different and require very different developments.

HYDRA appear from this need and its main objective is focused on facilitating this task, to minimize efforts and achieve more efficient results. That is why the system offers support for project management and provides a set of typical layout structures available to apply them.

On the other hand, HYDRA offers the possibility to process a project different formats for different output depending on the user's need. It also has defined a number of styles and templates that can be modified to adaptare any need for future generation.

Agradecimientos

Un mago nunca llega tarde, ni pronto. Llega exactamente cuando se lo propone, y por fin, ha llegado el momento de decir que llegué. Ha sido un etapa en la que a veces me sentía sola y frustrada, pero es en esos momentos en los que me he dado cuenta de quién estaba para sostenerme y ayudarme a ponerme en pie. Gracias a todos ellos es por quiénes he conseguido llegar aquí.

Elen sila lumen omontienvo

Índice general

Índice de cuadros

Índice de figuras

Índice de listados

Capítulo 1

Introducción

En los últimos siglos han ocurrido importantes avances tecnológicos que han propiciado enormes cambios en el modo de vida del ser humano. La publicación no es una excepción. Si bien es cierto que el hito más importante en este campo fue gracias a la invención de la imprenta moderna a mediados del siglo XV, en los últimos años han sucedido avances tecnológicos muy significativos en el ámbito de la edición digital.

Las ventajas que han proporcionado estos sistemas son cuantiosas, pero también presentan algunas dificultades. El que nos concierne para el presente proyecto es la heterogeneidad de formatos, lo que puede suponer que la tarea de obtener un mismo documento en distintos formatos sea tediosa y molesta para el usuario.

El proyecto HIDRA surge de la necesidad de crear un sistema que facilite la producción de archivos a distintos formatos. El objetivo es la definición de un lenguaje intuitivo para la definición de los distintos elementos que se deseen añadir a la publicación, y universal para todos los formatos de salida que contemple el sistema.

1.1 Sistemas de publicación

La historia no se entiende sin los registros por escrito de de todas las épocas. En la prehistoria, los hombres dejaban sus inscripciones talladas en la roca, si bien, la historia de la escritura no se iniciaría hasta miles de años después, estos constituyen los primeros antecedentes de la escritura.

En la Mesopotamia del IV milenio a.C. se encuentran los orígenes del que durante siglos ha sido la principal fuente de transmisión de información, el libro. En su origen, los primeros soportes fueron las tablillas de arcilla de los sumerios, haciendo uso de la escritura pictogramática, la cual también era usada por los egipcios, que la desarrollaron hasta usar la escritura a palabras y los rollos de papiro.

Tras el papiro llega el pergamino, aunque ambos conviven en Roma durante cientos de años. Si bien es cierto, que el formato de rollo fue dejando lugar a los *codex*, con un formato rectangular que recuerdan a los que actualmente conocemos como *libro*.

El pergamino no dejará de usarse hasta la llegada del papel, cuyos orígenes datan cerca

del año 150 d.C. procedentes de China. Las características del papel, que lo convertían en un material resistente y económico, facilitaron la difusión de las obras con la revolución que supuso la imprenta, en 1440 por Johannes Gutenberg. En este punto de la historia los escritos están soportados en libros tal y como los conocemos hoy en día.

En el año 1971 aparece el primer ebook, pero su difusión no se producirá hasta la década de los años 90, con la expansión de Internet.

La aparición del libro electrónico conllevó un importante cambio en las cadenas de producción y distribución de las publicaciones. La tendencia actual está obligando a los editores a cambiar las formas de publicación.

Como en cualquier evolución de un mercado incipiente, surgen diferentes aproximaciones tecnológicas que, unidas a intereses comerciales, ofrecen un amplio abanico de formatos con los que los editores de contenidos tienen que lidiar.

1.2 Impacto socio-económico

A pesar de la evolución de la tecnología, el grueso del negocio de publicación en España está aún en el formato físico. Aún así, se está produciendo un incremento lento pero continuado en la facturación de la publicación digital, a pesar de los reducidos ingresos del sector (en torno al 4 % total del facturación del libro digital en España, según el Observatorio de la Lectura y el Libro), que se encuentra en descenso desde el año 2009. La facturación global en 2013 sufrió una bajada del 11,7 %, en cambio, la del libro digital subió un 8,1 %. Por otro lado, los datos de Comercio Interior del Libro en España contó en 2013 con un total de 122.280 títulos digitales comercializados, un 123,5 % más que en el año 2012.

En relación a los dispositivos de lectura de las publicaciones digitales destaca el *streaming* o la lectura de documentos sin necesidad de descargarse en el dispositivos, por una cuota de suscripción, llegando a alcanzar casi el 50 % de la oferta editorial digital, en cambio la tendencia de compra de ordenadores, eReaders u otros dispositivos (para el consumo de libro digital) ha descendido notablemente en algunos casos como los dos primeros mencionados, el caso de móviles ha disminuido pero de manera poco notable. Esto queda reflejado en la tabla Comercialización del libro digital según dispositivos:

Dispositivos	***% 2011***	***% 2012***	***% 2013***	***% Variación***
Ordenadores	74,1	49,2	21,4	-27,8
EReaders	17,8	38,1	21,6	-16,6
Tabletas	1,8	5,2	6,1	0,8
Streamig/Online	2,3	4,4	48,6	44,3
Móviles/PDA	1,5	1,8	1,0	-0,8
Otros	2,5	1,3	1,3	0,0

Dispositivos	*% 2011*	*% 2012*	*% 2013*	*% Variación*

Cuadro 1.1: Comercialización del libro digital según dispositivos

A pesar de la idea que transmiten estos datos, de escasez de consumo digital, hay otros datos importante a tener en cuenta. Un artículo del periódico "El Mundo" con fecha del 6 de mayo del 2016 [1] indica que hay un 11,8 % de los españoles que lee en pantalla, y un 8,7 % que lee tanto en pantallas como en formato físico, en cambio sólo el 3 % de la cuota de mercado de las publicaciones es digital. El porcentaje que abarca desde el 3 % de la cuota de mercado, hasta el 20,5 de consumo digital incluye publicaciones de acceso gratuitas, pero también se deduce el gran impacto de la piratería en este sector.

1.3 Problemática

Hay un aumento de las publicaciones digitales que facilitan la distribución de las obras, aunque es importante indicar que en Internet no existe ninguna autoridad que controle la calidad de las publicaciones, pero el abaratamiento de costes, la rapidez de edición y distribución y la facilidad de acceso al contenido, son incentivos para los autores y usuarios finales.

Los formatos que presentan las distintas publicaciones abarca un amplio abanico de posibilidades. Si bien es cierto que algunos están más extendidos que otros, para llegar a un público lo más extenso posible es necesario trabajar con varios de ellos.

El Observatorio de la Lectura y el Libro, organismo perteneciente al Ministerio de Educación, Cultura y Deporte, realizó en abril de 2015 un estudio sobre el sector del libro entre los años 2013 y 2015[2].

Uno de los apartados que trató este estudio fue la edición y venta de libros digitales (capítulo 3). Respecto a los formatos de publicación, en el año 2014 la oferta en formato ePub representa el 41,2 %, y en formato PDF el 36,5 %. En la gráfica de la figura 1.1 obtenida del estudio del sector del libro del Observatorio de la Lectura y el Libro (referencia) se observa la producción de libros digitales según formatos en los años 2012, 2013 y 2014.

Uno de los principales problemas que detectan los usuarios con los lectores de libros electrónicos es que los formatos no son compatibles en todos los dispositivos. Kindle, por ejemplo, es el lector de Amazon, y sólo admite los archivos en formato MOBI y sus derivados, como azw. Otro ejemplo sería Tagus, que es el lector de Casa del Libro, que reconoce PDF y ePub. Nook por otra parte es incompatible con los archivos en formato MOBI.

Esta variedad de formatos para documentación digital escrita impulsa a los publicadores a

[1] http://www.elmundo.es/cultura/2016/05/06/572b9e47e2704ef35b8b456d.html
[2] Estimación obtenido de www.infojobs.net

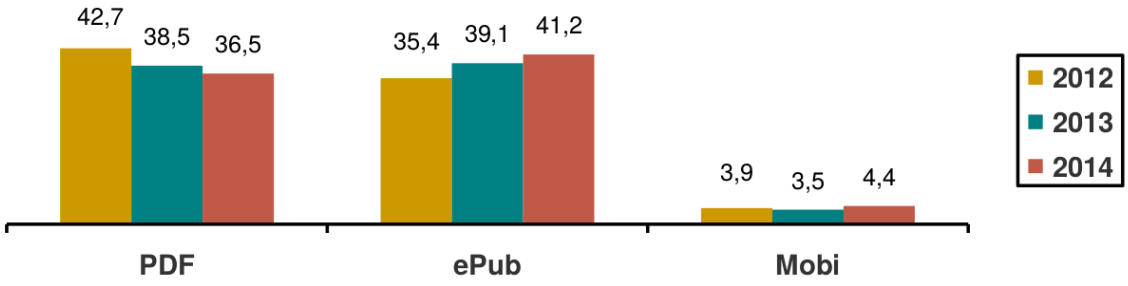


Figura 1.1: Producción de libros digitales según formatos

usar varios de estos formatos, para aumentar la accesibilidad de sus publicaciones. Todo lo anterior implica, tanto para el aprendizaje como para el uso de los lenguajes o tecnologías que generan los distintos soportes, un considerable esfuerzo, ya que dichos formatos emplean sintaxis diferentes y cuentan con requisitos técnicos muy heterogéneos; así como un elevado coste temporal y económico de la generación y maquetación de un mismo documento en distintos formatos.

Aunque en la actualidad existen algunas herramientas que permiten la transformación en distintos formatos (como el programa Calibre o Pandoc), resultan insuficientes cuando es necesaria la edición profesional de documentación técnica.

En el caso de Pandoc, es una herramienta por línea de órdenes que convierte documentos en una amplia cantidad de formatos, usando como base Markdown. Este, es un lenguaje de marcado ligero que evita la sobrecarga de otras propuestas orientadas a la producción de documentos como DocBook. Mediante Pandoc, un archivo fuente en formato Markdown se puede traducir a multitud de formatos.

Sin embargo, esta traducción automática está muy limitada a cierto número de características básicas que no cubren todas las necesidades de un entorno de edición técnica profesional.

HIDRA surge de esta necesidad, pretendiendo ser una alternativa a este tipo de herramientas, y enfocado especialmente a la edición documental de manuales científico-técnicos. a partir de un único formato, sencillo e intuitivo, genera distintos formatos de salida, atendiendo a la demanda actual de consumo.

HIDRA puede definirse como una herramienta de conversión automática que separa la definición del contenido de la parte de presentación, permitiendo a partir de un conjunto de archivos fuente (en formato Markdown enriquecido) y un archivo de descripción de proyecto, generar un conjunto de archivos de salida con plantillas personalizadas y contenido equivalente. En una primera etapa HIDRA soportará los siguientes formatos de salida, atendiendo a las necesidades docentes que se presentan actualmente en la UCLM: PDF (maquetado para impresión o publicación online, generado vía LaTeX), Epub2 (Ebook multidispositivo), Azw3 (Amazon Kindle) y SCORM3 (Versión 1.2, integración con Moodle).

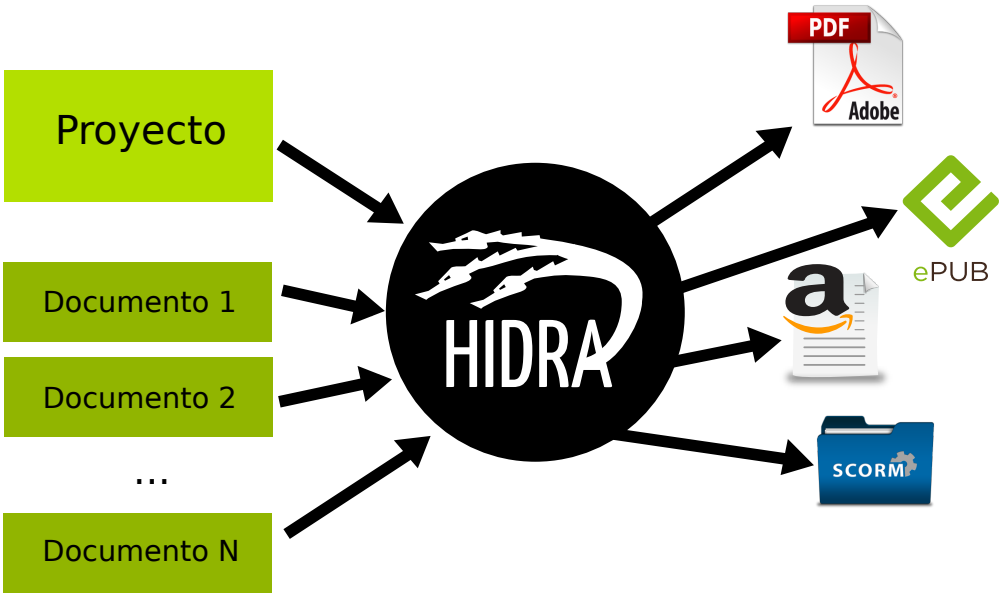


Figura 1.2: Sistema HIDRA

1.4 Estructura del documento

La normativa de trabajos de fin de grado de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha propone una estructura, que adaptándola al contenido del proyecto quedaría de la siguiente manera:

- **Capítulo 2: Objetivos**

Descripción del objetivo general del proyecto y de los objetivos específicos o subojetivos del sistema HIDRA.

- **Capítulo 3: Antecedentes**

En este capítulo se hace un estudio de los campos de investigación que intervienen de alguna forma en el desarrollo del sistema, y se realiza un estudio comparativo de las tecnologías a emplear para seleccionar la opción más viable en cada caso.

- **Capítulo 4: Método y fases de trabajo**

Capítulo en el que se describe la metodología de trabajo que se seguirá, detallando las fases de trabajo. También se describirán los medios empleados, tanto software como hardware, y las limitaciones que presenta el proyecto.

- **Capítulo 5: Resultados**

Se describe la evolución que ha sufrido el proyecto, detallando las iteraciones y pruebas experimentales de la productividad del sistema.

- **Capítulo 6: Arquitectura**

Se describirán los distintos módulos que componen HIDRA, siguiendo una perspectiva *top-down*, que muestra un enfoque general del sistema, y continuando con el análisis de los subsistemas de forma cada vez más exhaustiva, llegando a alcanzar en algunos casos aspectos detallados de la implementación.

- **Capítulo 7: Conclusiones**

En este capítulo se presentan las conclusiones que se han alcanzado tras la realización del proyecto, atendiendo también posibles mejoras y futuras líneas de aplicación.

Capítulo 2

Objetivos

En este capítulo se describe el objetivo general y los diferentes subobjetivos que se han planteado en el proyecto.

2.1 Objetivo general

El objetivo general del proyecto es el desarrollo de una herramienta de conversión automática enfocada a la edición documental de manuales científico-técnicos, la cual, proporcionando un proyecto con una estructura de ficheros definida, se traduzca a unos formatos de salida específicos que permitan la generación de libros digitales o su preparación para su posterior publicación en imprenta.

El proyecto HIDRA será utilizado como entorno para la generación de documentos y manuales en el Centro de Tecnologías y Contenidos Digitales (C:TED) de la Universidad de Castilla-La Mancha.

2.2 Objetivos específicos

A partir del objetivo general que ha sido descrito en la sección anterior, se definen los siguientes subobjetivos, que permitirán alcanzar el mismo:

2.2.1 Definición de un lenguaje propio HIDRA, basado en Markdown

Definición de aspectos específicos de un lenguaje que permita el procesamiento de etiquetas para los distintos elementos que se deseen maquetar. Al menos se dará soporte a los siguientes elementos y entornos:

- Capítulos y Secciones.
- Estilos básicos de texto (Normal, negrita, cursiva, subrayado y tachado).
- Notas a pie de página.
- Citas.
- Listas.
- Imágenes (Con soporte a la conversión automática de las imágenes adaptadas al formato de publicación), incluye imágenes simples, in line y laterales.

- Tablas.
- Ecuaciones.
- Cuadros de texto.
- Secciones de código.
- Enlaces.
- Letras capitales.
- Referencias.
- Bibliografía.
- Tags de Bajo Nivel (Posibilidad de incluir tags de bajo nivel específicos para cada formato de salida).

2.2.2 Definición de una estructura de proyecto

Los proyectos que se producirán con HIDRA deberán seguir una estructura de ficheros con los archivos involucrados, los estilos escogidos, así como los elementos que se deseen incluir, véase portada o bibliografía. Todo lo anterior será especificado en un archivo de proyecto.

2.2.3 Automatización de la conversión de elementos de edición

El sistema se encargará de convertir automáticamente los elementos de diseño empleados en el documento (tales como imágenes, ecuaciones y tablas) de modo que puedan visualizarse correctamente en un entorno multidispositivo.

2.2.4 Referencias Automáticas

HIDRA se encargará de la gestión de referencias automáticas entre imágenes, tablas, ecuaciones y elementos bibiliográficos en todos los formatos de salida.

2.2.5 Sistema multiplantilla

Se dará soporte a la creación de diferentes plantillas para cada uno de los formatos de salida soportados por el sistema. En un principio se han decidido definir como formatos de salida: PDF (maquetado para impresión o publicación online, generado vía LaTeX), Epub (Ebook multidispositivo), Azw3 (Amazon Kindle) y SCORM (Versión 1.2, integración con Moodle).

2.2.6 Soporte para código de bajo nivel

Se contempla la opción de proporcionar etiquetas para la inserción de código en un lenguaje concreto, como html o latex, por si se desea añadir alguna funcionalidad que no hubiera sido incorporada al sistema.

2.2.7 Portabilidad

El sistema deberá hace uso de tecnologías libres, multiplataforma. HIDRA debe ser ejecutado en GNU/Linux y Microsoft Windows.

2.2.8 Facilidad de uso

HIDRA está pensado para facilitar la tarea de portar a distintos formatos, pero además el lenguaje definido para ello será muy intuitivo y cómodo de usar. Se incluirá un manual de usuario con todas las posibilidades que ofrece el sistema.

Capítulo 3

Antecedentes

Los objetivos propuestos en la sección Objetivos requieren la adquisición de conocimientos de muy diversas áreas de la informática. En el presente capítulo se ha pretendido poner de manifiesto los conceptos que se consideran básicos en el desarrollo del trabajo de fin de grado, agrupados en secciones. En HIDRA intervienen distintas áreas, como el procesamiento del lenguaje, incluyendo el estudio de lenguajes, gramáticas, expresiones regulares, funcionamiento de los compiladores; el estudio de las tecnologías a las que irá destinado el resultado de HIDRA, como libros electrónicos o campus virtual; la definición de lenguajes propios y el estudio de herramientas o técnicas similares a lo que se pretende obtener con HIDRA.

En la figura 3.1 se observa un mapa conceptual con los conceptos que se describen en el presente capítulo.

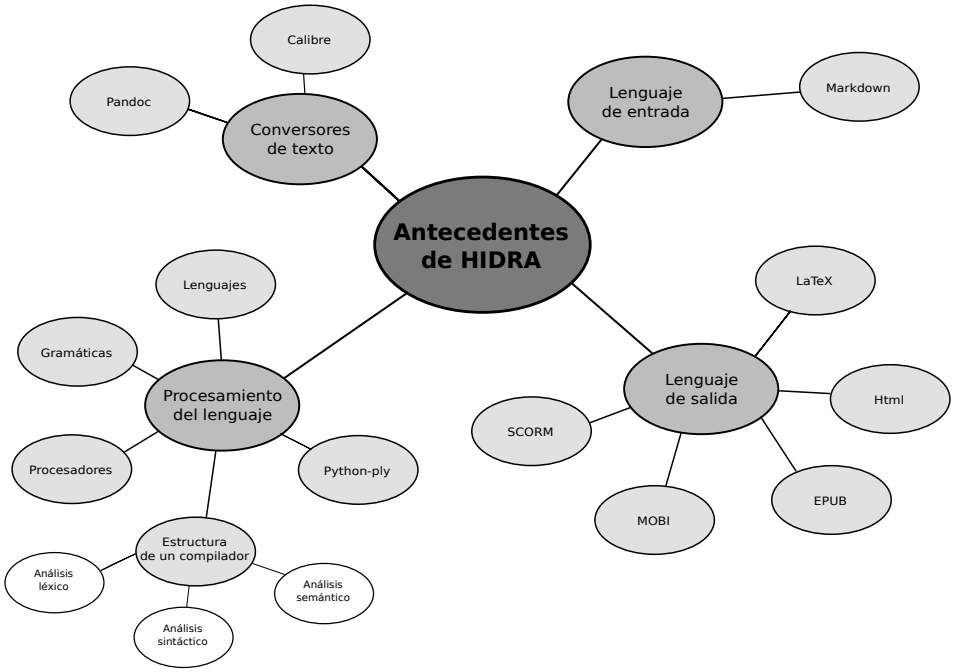


Figura 3.1: Mapa conceptual de los antecedentes

3.1 Conversores de texto

En esta sección se repasan distintas herramientas que cumplen algunos de los objetivos de HIDRA, con algunas de las funcionalidades que se desean para el sistema.

3.1.1 Pandoc

Pandoc [12] es un conversor universal de documentos, además es libre y de código abierto. Fue desarrollado por John MacFarlane, profesor de filosofía en la Universidad de California. Además cuenta con un módulo, denominado *Pandoc-citeproc* que se encarga de la gestión de referencias bibliográficas. Además, es una herramienta multiplataforma y fácil de usar. Para HIDRA se ha decidido hacer uso de algunas de las funcionalidades que proporciona, estás están explicadas en el Módulo de conversión de documentos.

A continuación se muestra un ejemplo de uso de pandoc:

```
pandoc file.html --highlight-style=pygments --webtex file.epub
```

Código 3.1: Prueba de uso de Pandoc

En el ejemplo, la opción *–highlight-style=pygments* indica el estilo de color que se usa para resaltar la sintaxis, y *–webtex* renderiza las fórmulas haciendo uso de un script externo a Pandoc [13].

Puede convertir documentos en markdown, reStructuredText, textile, HTML, DocBook, LaTeX, MediaWiki markup, TWiki markup, OPML, Emacs Org-Mode, Txt2Tags, Microsoft Word docx, LibreOffice ODT, EPUB, o Haddock markup a HTML, docs, odt, xml, EPUB, DocBook, TEI Simple, GNU TexInfo, Groff man pages, Haddock markup, InDesign ICML, OPML, LaTeX, ConTeXt, PDF, Markdown, reStructuredText, AsciiDoc, MediaWiki markup, DokuWiki markup, Emacs Org-Mode y Textile.

3.1.2 Calibre

Calibre [14] es un software multiplataforma y libre, implementado en Python y C. Es un gestor de libros electrónicos, pero también permite la conversión a formatos de libro electrónico, y es por esta razón por la que se realizó el estudio de esta herramienta.

En la figura 3.3 se muestra la pantalla principal del programa con un libro de ejemplo.

A continuación, se muestra la conversión del libro de ejemplo, que está en formato PDF a formato ePub de manera gráfica. Figura 3.4

También es posible la conversión de libros usando comandos, la equivalencia de la transformación gráfica de la figura 3.4 por línea de comandos sería:

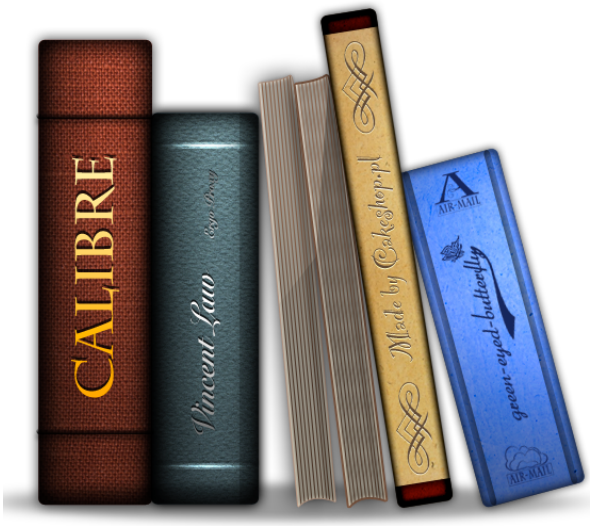


Figura 3.2: Pantalla de arranque de Calibre

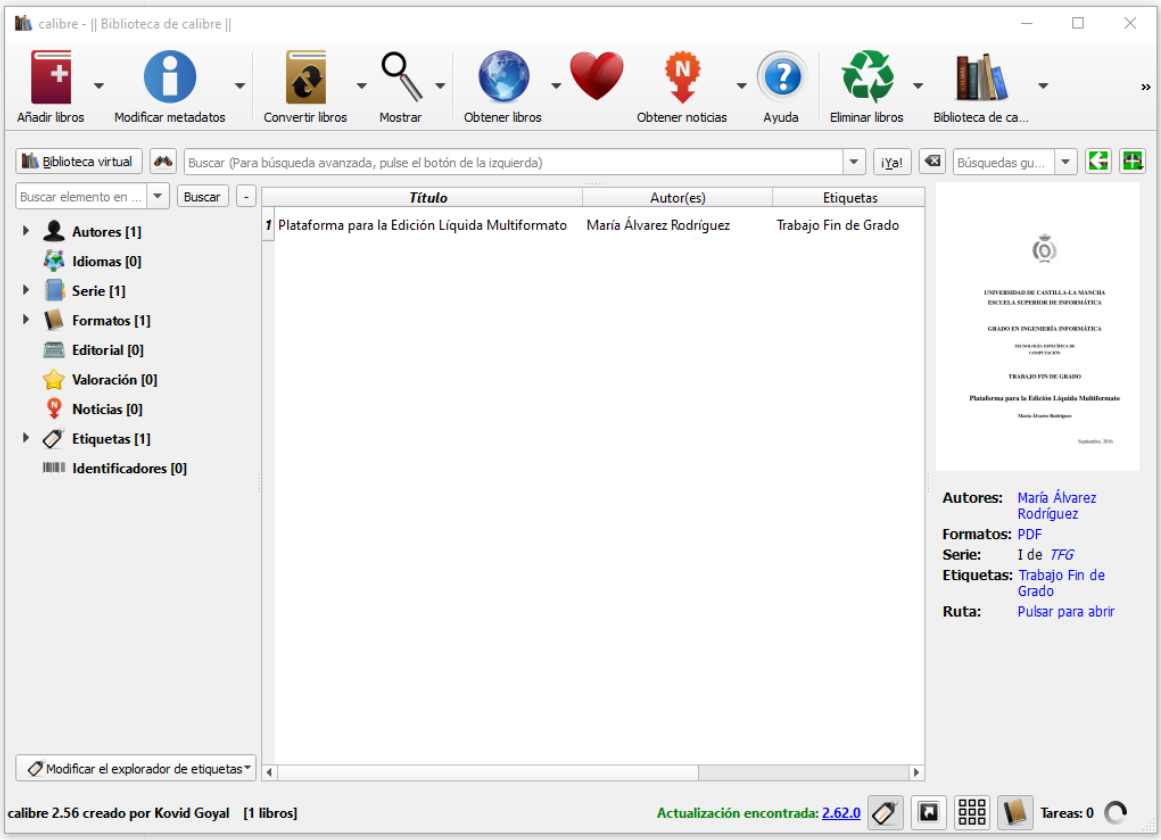


Figura 3.3: Pantalla principal de Calibre

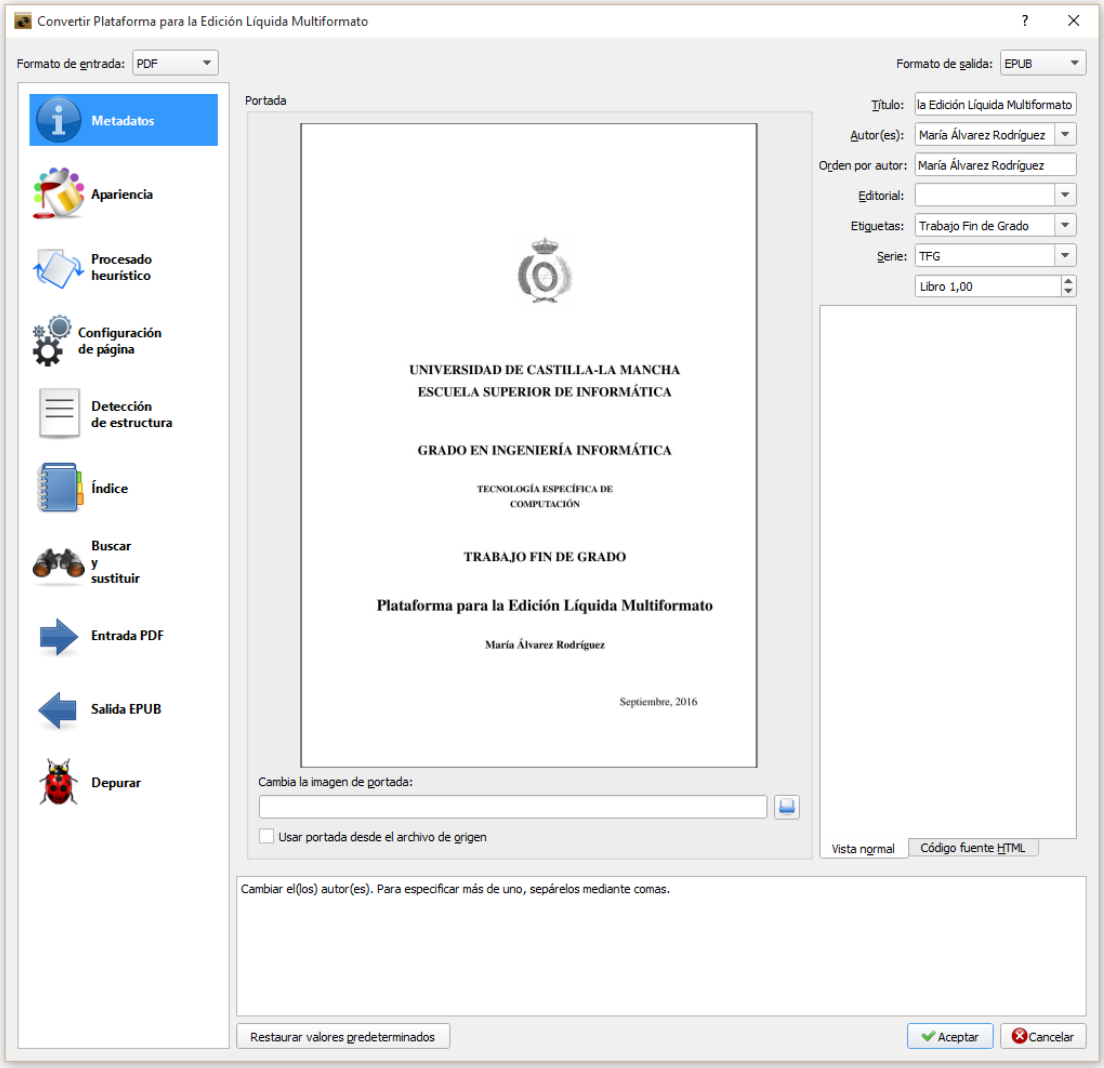


Figura 3.4: Conversión de un libro de forma manual

```
ebook-convert file.pdf file.epub --read-metadata-from-opf metadata.txt
```

Código 3.2: Conversión de libros por medio de comandos

Donde *metadata.txt* es un archivo que contiene la metainformación relativa al libro.

Puede convertir documentos en ePubm HTML, PDF, RTF, txt, cbc, fb2, lit, MOBI, ODT, prc, pdb, PML, RB, cbz y cbr a ePub, fb2, OEB, lit, lrf, MOBI, pdb, pml, rb.

3.2 Procesamiento del lenguaje

En esta sección se introducirán los conceptos básicos del procesamiento del lenguaje, incluyendo los lenguajes y las gramáticas formales, tipos de procesadores, y la estructura básica de un compilador, además se realizará una breve descripción de la notación BNF (Backus-Naur), mostrando como ejemplo la gramática empleada en HIDRA; y por último se hará especial incapié en las herramientas proporcionadas por la biblioteca python-ply, una implementación de las herramientas lex y yacc para análisis léxico y sintáctico, desarrollado en python.

3.2.1 Lenguajes

Citando a [1], un **lenguaje** es un conjunto finito o infinito de oraciones, cada una de ellas de longitud finita y construidas por concatenación a partir de un número finito de elementos.

Formalmente, y con respecto a la definición formal de gramática que se muestra en la siguiente subsección Gramáticas formales, el lenguaje generado por una gramática:

$$G = (\Sigma_N, \Sigma_T, S, P) \tag{3.1}$$

será notado como *L(G)*, y se define como el conjunto de cadenas formadas por símbolos terminales que son derivables a partir del símbolo inicial de la gramática.

$$L(G) = \{u\varepsilon\Sigma_T | S^* \Rightarrow u\} \tag{3.2}$$

Chomsky realizó una clasificación tanto de las gramáticas como de los lenguajes formales, dividiéndolos en cuatro tipos [11], siendo *L* el lenguaje definido por la gramática *G*:

- *Lenguajes regulares: (Tipo 3)* Los lenguajes regulares pueden ser definidos por un autómata finito, y se utilizan para definir el léxico de los lenguajes de programación.

- *Lenguajes independientes de contexto: (Tipo 2)* Son los generados por las gramáticas libres o independientes de contexto.
- *Lenguajes sensibles (o dependientes) al contexto: (Tipo 1)* Las gramáticas de Tipo 1 generan este tipo de lenguajes. Ver la sección Gramáticas formales para más detalles.
- *Lenguajes sin restricciones: (Tipo 0)* La gramática de L (o gramáticas) no cumple ninguno de los casos anteriores.

La relación que existe entre los lenguajes que acaban de definirse es la que se muestra en la figura 3.5:

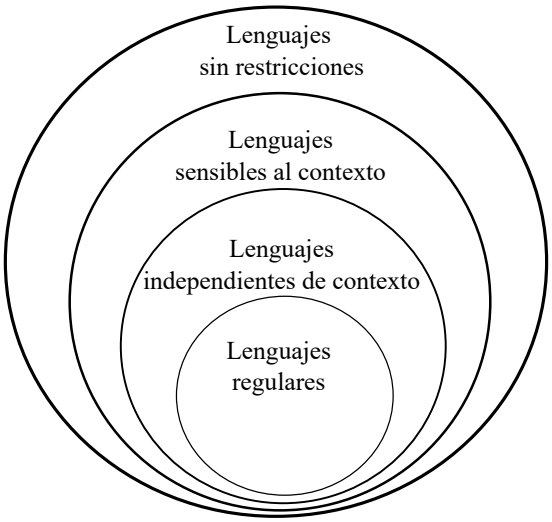


Figura 3.5: Clasificación de los lenguajes de Chomsky

Siguiendo con la definición del lenguaje, son necesarios tres aspectos fundamentales:

- **Léxico:** Constituido por el conjunto de palabras que pertenecen al lenguaje, y sus respectivas categorías.
- **Sintáctico:** Son las reglas que definen como se construyen oraciones o frases válidas del lenguaje, haciendo uso de los elementos que pertenecen al vocabulario.
- **Semántico:** Define el significado de las frases.

Derivación

En esta sección se realiza una breve descripción del concepto de derivación.

Dada la gramática

$$G = (\Sigma_N, \Sigma_T, S, P) \tag{3.3}$$

y sean $\alpha, \beta\varepsilon(\Sigma_N \cup \Sigma_T)^*$ dos palabras, se dice que β es derivable a partir de α en un paso $(\alpha \Rightarrow \beta)$ si y sólo si existen palabras $\delta_1, \delta_2\varepsilon(\Sigma_N \cup \Sigma_T)^*$ y una producción $\gamma \rightarrow \varphi$ tales que $\alpha = \delta_1\gamma\delta_2$ y $\beta = \delta_1\varphi\delta_2$.

Con los mismos supuestos, se dice que β es derivable a partir de $\alpha(\alpha^* \Rightarrow \beta)$ si y solo si existe una sucesión de palabras $\delta_1, \delta_2, ..., \delta_n(n \geq 1)$ tales que $\alpha = \delta_1 \Rightarrow \delta_2 \Rightarrow ... \Rightarrow \delta_n = \beta$.

3.2.2 Gramáticas formales

Las gramáticas describen la estructura de las palabras de un lenguaje. Por lo tanto, se usan para generar todas las palabras pertenecientes a un lenguaje dado.

Formalmente, se denomina *gramática formal* a una cuádrupla:

$$G = (\Sigma_N, \Sigma_T, S, P) \tag{3.4}$$

donde:

- Σ_N, es un conjunto de símbolos no terminales.
- Σ_T, es un conjunto de símbolos terminales.
- *P* es un conjunto de pares (α, β), denominadas **reglas de producción**, donde $\alpha, \beta\varepsilon(\Sigma_N \cup \Sigma_T)^*$ y α contiene al menos un símbolo de Σ_N. Su representación es $\alpha \rightarrow \beta$.
- *S* es un elemento de Σ_N, llamado **símbolo inicial**.

Como se mencionó anteriormente Chomsky realizó una clasificación de las gramáticas, dividiéndolas, al igual que los lenguajes, en *Tipo 0, Tipo 1, Tipo 2, Tipo 3*, entre los que existe la siguiente relación de inclusión:

$$Tipo3 \subset Tipo2 \subset Tipo1 \subset Tipo0 \tag{3.5}$$

- *Gramáticas regulares: (Tipo 3)* Son las gramáticas que generan los lenguajes regulares. Las producciones de G tienen la forma $A \rightarrow x$ ó $B \rightarrow x$, dónde *A* y *B* son símbolos no terminales, y *x* es un terminal.
- *Gramáticas independientes de contexto: (Tipo 2)* Todas las producciones de G tienen la forma $A \rightarrow x$, dónde *x* es una combinación de símbolos terminales y no terminales.
- *Gramáticas sensibles (o dependientes) al contexto: (Tipo 1)* Las producciones de G son de las forma $xAy \rightarrow xzy$, donde *A* es un símbolo no terminal, y *x,y,z* son combinaciones de terminales y no terminales, pudiendo ser *x,y* cadenas vacías.

- *Gramáticas sin restricciones: (Tipo 0)* No cumple ninguno de los casos anteriores.

En la siguiente tabla se observa la relación entre lenguajes, gramáticas y autómatas:

Gramáticas	Lenguajes	Autómatas
Tipo 2	Independientes de contexto	Autómata a pilas
Tipo 3	Regulares	Autómata finito

Cuadro 3.1: Relación entre lenguajes, gramáticas y Autómatas

3.2.3 Procesadores del lenguaje

Un procesador es una aplicación que a partir de unos archivos descritos en un lenguaje, conocido como lenguaje fuente.

Hay distintos tipos de procesadores del lenguaje: compiladores, intérpretes, preprocesadores, ensambladores, enlazadores, cargadores, desensambladores, decompiladores, analizadores de rendimiento, optimizadores de código, compresores, formateadores, editores...

- **Traductor**: tipo de procesador, que procesa los archivos de entrada, descritos en lenguaje fuente, en otro lenguaje, conocido como lenguaje objeto. Está escrito en un lenguaje de implementación. Por lo tanto, en el proceso de traducción intervienen tres lenguajes distintos.
- **Compilador**: es un tipo concreto de traductor. El lenguaje de entrada es de alto nivel, y el de salida es de bajo nivel. Para la ejecución del programa descrito en lenguaje fuente que necesita hacer uso de un compilador se deben tener en cuenta dos tiempos: El tiempo de compilación, que es el tiempo necesario para traducir del lenguaje de alto nivel al lenguaje de bajo nivel; y el tiempo de ejecución, que es el tiempo necesario para ejecutar el código en lenguaje objeto.
- **Enlazadores**: El proceso de enlazado tiene lugar entre la compilación y la ejecución, y ocurre cuando los programas se han compilado por separado. El enlazador se encarga de crear un módulo de carga (que es el programa completo en lenguaje objeto).
- **Cargadores**: Es el encargado pasar el control a la primera instrucción del lenguaje objeto, para comenzar la ejecución. También se encarga de la asignación del espacio necesario en memoria.
- **Preprocesador**: tipo de compilador encargado de tareas como incluir ficheros o ampliar macros.
- **Ensamblador**: tipo de traductor cuyo lenguaje fuente es ensamblador, y lenguaje objeto es código máquina. Por regla general, el lenguaje fuente tiene una estructura sencilla, se traduce una sentencia fuente a una sentencia en lenguaje máquina. Los ensam-

bladores que cuentan con macroinstrucciones en su lenguaje se denominan macroensambladores.

- **Intérprete**: tipo de procesador del lenguaje que analiza los archivos descritos en lenguaje fuente y lo ejecuta sin generar código objeto. En este caso todo se realiza en tiempo de ejecución
- **Emulador**: es un tipo de intérprete, donde los archivos de entrada están descritos en el código máquina de otra plataforma distinta a la que la está interpretando.

3.2.4 Estructura de un compilador

La estructura de un compilador puede dividirse en distintas fases, entre las que no existe una total independecia, de hecho algunos módulos de dichas fases se solapan.

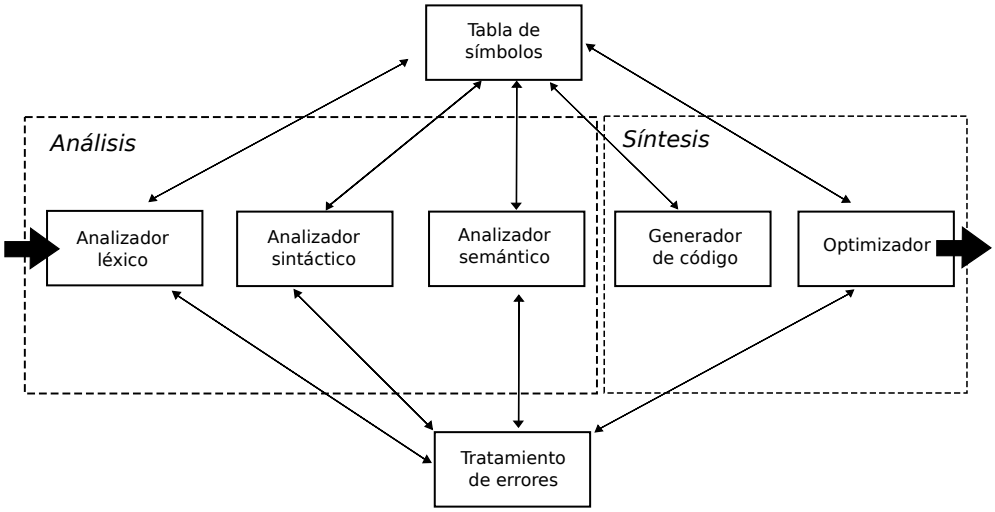


Figura 3.6: Fases de un compilador

- ***Fase de análisis***. Consiste en comprobar que el lenguaje fuente es correcto, para ello, esta fase se divide en tres etapas:
  - **Análisis léxico**: Descompone el lenguaje fuente en unidades mínimas denominadas *tokens*, y se comprueba si están construidas correctamente.
  - **Análisis sintáctico**: Se comprueba si las agrupaciones de tokens pueden disponerse correctamente en alguna de las sentencias del lenguaje a analizar.
  - **Análisis semántico**: En esta fase se verifica la consistencia de tipos, definición de variables... Y se genera un árbol.
- ***Fase de síntesis***. El objetivo es obtener el código en lenguaje objeto. Es común, en el caso de los compiladores generar el código intermedio, que corresponde a una máquina abstracta, para que sea utilizado en otras máquinas sin que sea necesario volver a realizar la fase de análisis.

- **Generación de código**: A partir del código intermedio se genera el código máquina.
- **Optimización de código**: Se realiza una serie de transformaciones para obtener el código objeto lo más eficientemente posible.

3.2.5 Python-ply

Python-ply es una implementación de las herramientas lex y yacc para análisis léxico y sintáctico, desarrollado en python, fue creado por David Beazley en el 2001, en una clase de *Introducción a los compiladores* de la Universidad de Chicago.

Según [15] presenta las siguientes características:

- Está implementado completamente en **Python**.
- Hace uso de un **analizador sintáctico LR**, lo que implica que, además de ser un analizador ascendente, que hace uso de la técnica de análisis por reducción.
- **Python-ply** incluye algunas características claves del análisis léxico y sintáctico, ya implementadas, como son las *producciones nulas*, *reglas de prioridad*, *recuperación de errores* y *soporte para gramáticas ambiguas*.
- Facilidad de uso y una amplia comprobación de errores.
- Únicamente proporciona la funcionalidad de analizador léxico y sintáctico.

Está constituido por dos módulos separados, por un lado *lex.py* y por otro *yacc.py*. *Lex.py* es el analizador léxico, que define el vocabulario del lenguaje con un conjunto de expresiones regulares. *Yacc.py*, necesario para reconocer la sintaxis del lenguaje definida mediante una gramática libre de contexto.

Ambas herramientas trabajan de forma cooperativa: *lex.py* proporciona a *yacc.py* el siguiente token (entendido como símbolo terminal de la gramática) válido del flujo de entrada. *Yacc.py* con estos tokens invoca las reglas gramaticales. La salida por defecto es un árbol sintáctico, pero puede ser configurable por el usuario.

Entrando más en detalle, *lex.py* divide una entrada dada, en tokens individuales, indicando su tipo y su valor. Para conseguirlo, los diferentes tipos de tokens se expresan mediante expresiones regulares. En la sección 6.4 se muestra la nomenclatura empleada con ejemplos de léxico de HIDRA.

3.3 Lenguaje de entrada

Por experiencia, se sabe que no es especialmente cómodo para el usuario tener que aprender sintaxis distintas para cada aplicación que se desee usar. Por esta razón se decidió buscar un lenguaje estandarizado y fácil de usar, ya que posteriormente deberíamos añadir nuestros propios elementos. Se optó por el uso de *Markdown*.

3.3.1 Markdown

Markdown [16] es un lenguaje de marcado ligero, lo que significa que está estandarizado y que es fácil de editar. Su origen proviene de intentar convertir texto plano a html, esto ocurrió en 2004, por John Gruber. Entre algunas de sus características está la opción de añadir formatos utilizando texto plano. En el apartado "Sintaxis de markdown" se muestra un pequeño manual de uso de los ejemplos más significativos.

Otro punto destacable de Markdown es que es compatible con todas las plataformas.

3.4 Lenguajes de salida

Una de las características deseables en el sistema HIDRA es la accesibilidad de sus proyectos. La facilidad de "traducir" una publicación en distintos formatos es clave, pero ¿Cuáles son los que más convienen a este tipo de sistemas? Si se enfoca desde un punto de vista educativo, tener un buen formato en papel es primordial, pero no es el único. Con el avance de las nuevas tecnologías, la posibilidad de estudiar de forma digital ha abierto un mundo de posibilidades, luego, el formato de libro electrónico se considera también imprescindible. Por último, se estudió el formato SCORM, que es el que se usa en el campus virtual, con el objetivo de ser introducido también como material evaluable en moodle.

El sistema HIDRA proporciona la posibilidad de obtener el proyecto de entrada en distintos formatos de salida, actualmente están implementados los siguientes:

- PDF
- Html
- EPUB
- MOBI
- SCORM

Se realizó un estudio de cada uno de ellos, algunos como SCORM, con mayor profundidad, para conocer los elementos necesarios para la implementación del sistema.

3.4.1 LaTeX

Si bien es cierto que el formato de salida que proporciona HIDRA es PDF, esto es por medio de LaTeX, que según [17] es un sistema de preparación de documentos para la composición tipográfica de alta calidad, por lo que se procedió al estudio de los distintos elementos que se consideraron imprescindibles para el proyecto, como la inserción de imágenes, ecuaciones o secciones de código fuente, entre otros.

Se optó por definir funciones propias en latex para las distintas etiquetas disponibles. Además, se han definido o adoptado distintos estilos disponibles para aplicar con HIDRA, aunque está disponible la edición de estos estilos al gusto del usuario.

3.4.2 Html

El estudio que se realizó de *html* [19] realmente fue breve, y consistió en la decisión de los formatos de salida derivados de *html* que se iban a implementar. Aunque también se proporciona si el usuario lo desea, la salida en formato html, desde el desarrollo se trató como un lenguaje intermediario, para facilitar la obtención de otros formatos como epub, mobi o scorm, cuenta con una hoja de estilos muy simple, también editable por parte del usuario.

3.4.3 EPUB

EPUB (Electronic Publication) [18] es un formato redimensionable de código abierto para leer textos e imágenes, se trata de un estándar, por lo que es soportado por la mayoría de los lectores de libros electrónicos de la actualidad, a excepción de Kindle, (razón por la que se decidió incluir también el formato MOBI en HIDRA), además usa notificación *Unicode* [20]. El estudio que se realizó del formato EPUB se basó en el análisis de la estructura que constituye este tipo de archivos. Dicha estructura consiste en una serie de archivos con formato *xhtml* que constituirían los capítulos del libro, además, dichos archivos pueden tener asociados el estilo (archivo css) e imágenes, que deben ser incluidos en el *.epub*. También debe incluir dos archivos que juntos determinan la estructura del libro electrónico. Estos archivos son:

- Archivo con extensión *.opf*, contiene toda la metainformación del libro, incluyendo título, autor, editorial... pasando los elementos multimedia, los capítulos y las referencias.

- Archivo con extensión *.nxc*, define la tabla de contenidos del libro electrónico.

Todos los archivos descritos anteriormente deben ir comprimidos en un contenedor con el formato de salida *.epub*.

3.4.4 Mobi

La decisión de implementar las operaciones necesarias para obtener la salida también en formato *.mobi* se debe a que los lectores de libros electrónicos Kindle de Amazon no soporta el formato *epub*. Es un formato de libros electrónicos creado por *Mobipocket S.A.* [21], en el que se define el contenido pero no se delimita el formato, para adaptarlo a distintos dispositivos. Si bien está basado en las especificaciones de Open eBook (Open eBook Publication Structure, formato de libro digital basado en el lenguaje xml), la edición de este tipo de formato no está disponible directamente, por lo que la alternativa disponible era la transformación del archivo con formato *.epub* a *.mobi*.

3.4.5 Scorm

Citando a [22] un paquete SCORM es un bloque de material web empaquetado de una manera que sigue el estándar SCORM de objetos de aprendizaje. Por lo tanto hay que distinguir dos conceptos, por un lado SCORM, entendido como un conjunto de estándares y especificaciones que permite crear objetos pedagógicos estructurados, y por otro lado el paquete que sigue dichos estándares.

Capítulo 4

Método y fases de trabajo

La elección de una apropiada metodología de desarrollo resulta imprescindible para alcanzar los objetivos propuestos en el capítulo 2 lo más eficientemente posible.

En esta decisión intervienen distintos aspectos como presupuesto, dimensiones del proyecto, tiempos de entrega, personal, recursos disponibles, entre otros.

En el presente capítulo se detallará la metodología de desarrollo escogida, así como un listado de las herramientas empleadas.

4.1 Metodología de desarrollo

HIDRA es un proyecto en el que trabajan varios desarrolladores, y en el que se van añadiendo nuevas funcionalidades cada poco tiempo, por lo tanto es necesaria una metodología de desarrollo que se adapte a estas necesidades.

Los componentes del equipo de desarrollo son:

- María Álvarez Rodríguez (Coordinadora del proyecto)
- Alberto Izquierdo Ramírez
- Santiago Sánchez Sobrino
- Y por último, Pedro Manuel Gómez-Portillo López, que se incorporó en las últimas semanas de desarrollo.

Inicialmente, tanto Santiago y Alberto, como yo, nos distribuimos las tareas de investigación de las tecnologías a usar, para especializarnos en alguna de ellas, y aunque como *project leader* he participado en todas las áreas de desarrollo, a grandes rasgos la distribución quedaría de la siguiente manera: Santiago se encargó del módulo de generación del proyecto, encargándose de las tareas de interpretación del archivo *Proyecto.hip*, el módulo de *keywords* y el módulo de adaptadores; Alberto, por su parte, se encargó del módulo de entrada y salida y en parte del módulo de traducción; Pedro se encargó del módulo de traducción, y yo, de los módulos de preprocesamiento, *parser* o procesamiento del lenguaje y traducción. Vuelvo a reiterar el hecho de que no se realizó una implementación tan cerrada, y que todos participamos en la mayoría de las tareas en mayor o menor medida.

Al comenzar el proyecto se seleccionó como la metodología de desarrollo a emplear el proceso iterativo e incremental. Esta decisión se tomó por dos cuestiones, la primera, el método en cascada no cumplía los suficientes requisitos, ya que al comienzo del proyecto no se tenía claro el alcance del sistema y las funcionalidades del mismo; y la segunda, si bien el equipo estuvo formado por tres e incluso 4 personas en algunas ocasiones, la metodología SCRUM podría resultar contraproducente; se decidió que la tercera opción a discutir podría cumplir los requisitos para el desarrollo de HIDRA, y así pues comenzamos el desarrollo aplicando el método iterativo e incremental.

En el capítulo 6 se describe la arquitectura del sistema, que se dividió en módulos. En cada iteración se fueron ampliando dichos módulos, y al final de cada una de ellas se entregaba un prototipo funcional, permitiendo evaluar los avances y definir nuevos requisitos.

En el esquema de la figura 4.1 puede observarse el esquema que se suele seguir en la metodología iterativa incremental.

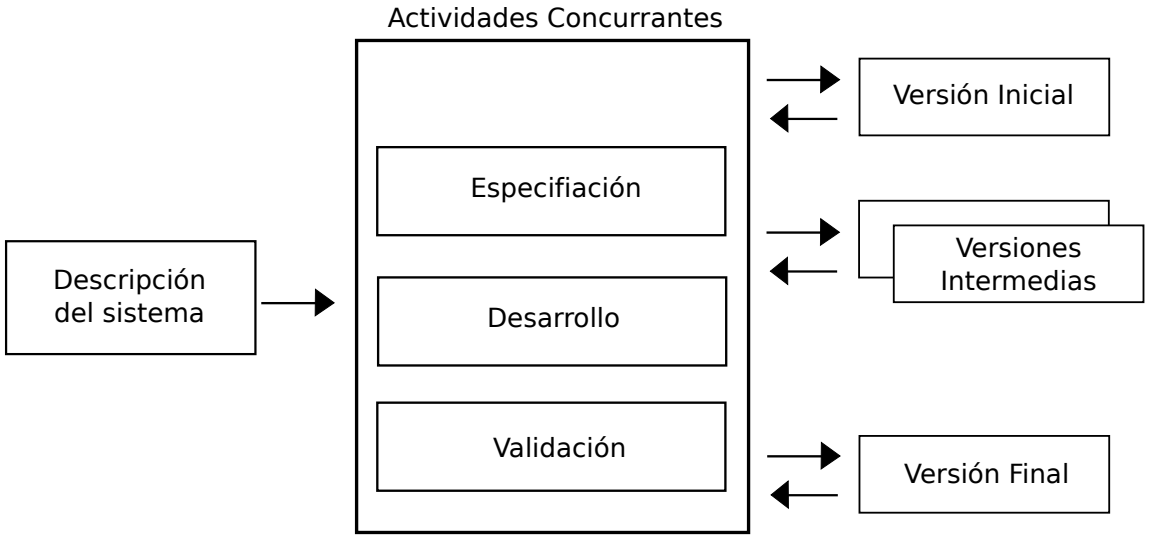


Figura 4.1: Esquema metodología iterativa incremental

La metodología iterativa incremental [7] se caracteriza por planificarse en bloques temporales que denominamos iteraciones, agregando más opciones de requisitos y logrando así una mejora mucho más completa.

La actividad de esta metodología puede dividirse en:

- **Etapa de inicialización**, en la que se realiza un análisis de los requisitos del sistema y las nuevas funcionalidades.
- **Etapa de iteración**, en la cual se realizan cambios en el diseño y se agregan nuevas funcionalidades
- **Etapa de pruebas y documentación**, para obtener al terminar una iteración una ver-

sión final del software.

4.2 Herramientas empleadas

Para el desarrollo de HIDRA se ha hecho uso de herramientas tanto hardware como software, a continuación se detallan estas herramientas, con su versión específica y una pequeña descripción.

4.2.1 Hardware

Los medios hardware que serán necesarios para el desarrollo del proyecto:

- Computador ASUS A55A: para el desarrollo del proyecto.
- Bq - E-reader cervantes: que permita hacer las pruebas pertinentes con respecto al desarrollo del formato de salida EPUB.
- Libro electrónico Kindle Paperwhite: que permita hacer las pruebas pertinentes con respecto al desarrollo del formato de salida MOBI.

4.2.2 Software

Los medios software que serán necesarios para el desarrollo del proyecto:

- Windows 7 [24], GNU/Linux [25] Ubuntu: El computador contará con ambos sistemas para asegurar el correcto funcionamiento del proyecto en ambos.
- Moodle 2.99 [28]: Plataforma de aprendizaje diseñado para permitir la creación de entornos de aprendizaje personalizados.
- Pandoc 1.12.2.1 [12]: El software de código abierto Pandoc, que ya se menciono en la introducción, se empleará en una transformación primitiva ,que sufrirán los ficheros del proyecto generado por HIDRA, antes de ser procesados por el sistema.
- Calibre 1.25 [14]: Gestor de libros electrónicos libre, que permite la conversión entre formatos de archivos para libros electrónicos.
- ImageMagick 6.9.2 [32]: Software de código libre que permite la manipulación de imágenes y un procesamiento adecuado para obtener el formato deseado que precise HIDRA.
- Latex [17]: Genera el maquetado para impresión o publicación online.
- Bibtex2html [33]: Colección de herramientas que permite la traducción de bibtex (formato que se empleará para la bibliografía) a HTML.
- Tex2im [34]: Herramienta que traduce fórmulas en formato Latex a imágenes.
- Reload SCORM Player v1.2 [35]: Programa de código libre que permite la creación de paquetes SCORM.

4.2.3 Herramientas de desarrollo

- Git 2.6.2 [26]: El proyecto será mantenido mediante control de versiones a través de un repositorio online.
- Trello [27]: Sistema web que permite la gestión de proyectos.

4.2.4 Lenguajes de programación

- Python 2.7 [29]: Lenguaje principal en el que se desarrollará el sistema.
- HTML [19]: Lenguaje necesario para la producción de los formatos EPUB, MOBI y SCORM.
- JavaScript [30]: Para los aspectos relacionados con la producción de archivos de SCORM.

4.2.5 Bibliotecas

- Gtk [31]: Conjunto de bibliotecas multiplataforma para el desarrollo interfaces gráficas de usuario.
- Python-ply [29]: Biclioteca que permite realizar un análisis léxico y sintáctico.

Capítulo 5

Resultados

En el presente capítulo se diferencias dos partes, por un lado se describe la evolución sufrida por el proyecto durante su etapa de desarrollo, detallando las iteraciones que se han realizado para su finalización, especificando las tareas llevadas a cabo en cada una de ellas; y por otro lado, los costes asociados al proyecto, con una estimación tanto económica como temporal, y se detalla el proceso de un ejemplo de uso real.

5.1 Evolución del proyecto

Con el inicio del proyecto se realizó un análisis de los requisitos del proyecto. Si bien es cierto, que desde dicho comienzo hasta la actualidad, el sistema ha sufrido muchos cambios, los aspectos fundamentales no han variado, permitiendo así realizar una implementación basada en el diseño inicial. A continuación se describe brevemente el análisis que se realizo inicialmente, y posteriormente se detallan las iteraciones que tuvieron lugar en la elaboración del software.

5.1.1 Análisis de requisitos

El equipo de trabajo comenzó siendo formado por tres integrantes, que recibimos una especificación para el desarrollo del sistema HIDRA, dicha especificación inicial puede verse en el Anexo A: Requisitos.

> El **objetivo** era proporcionar una herramienta de conversión automática que, partiendo de un documento especificado en un formato tipo Markdown, obtenga una salida en diversos formatos para publicación. En principio, se plantea el soporte para los siguientes tipos de archivo de salida: PDF (maquetado para impresión o publicación online, generado vía LaTeX), Epub (Ebook multidispositivo), Mobi (Amazon Kindle), Scorm (integración con Moodle).
>
> **Resumen de Especificación Técnica:** El sistema contará con un archivo de especificación de proyecto y un árbol de directorios donde se especificarán los recursos gráficos y de texto a utilizar. El sistema tomará como entrada ese archivo y generará los documentos de salida.

Se acompañaba también de un ejemplo de archivo de proyecto, y una lista de las etiquetas que el sistema debía soportar como mínimo. (Véase Anexo A: Requisitos).

5.1.2 Iteraciones

A continuación se describen las iteraciones seguidas durante el desarrollo del proyecto.

Iteración 1

En esta primera etapa se llevó a cabo una investigación de las tecnologías que en un inicio se consideraron necesarias. Entre ellas Pandoc, los formatos ePub y MOBI interactivos, script con LUA y SCORM.

Además se generaron ejemplos de prueba de algunos de ellos para probar su funcionamiento, destacar el caso de SCORM para mencionar el uso de *RELOAD Project: Editor* herramienta para la validación de paquetes SCORM.

A continuación se comenzó la implementación de la estructura básica de la arquitectura de HIDRA. Al motor principal lo denominamos HidraCore, y posteriormente desaparecería, derivando en otras clases, más adelante se detallan.

Se crearon las clases de las primeras etiquetas y se realizó la implementación de la API para la gestión de proyectos y su interfaz pública.

Otras de las tareas que se llevaron a cabo fueron la creación de un diccionario de etiquetas, la lectura y el preprocesado de los HidraDocuments (*.hid*), que finalmente serán archivos *Markdown* (*.md*), se permitió el orden desordenado de los atributos dentro de las etiquetas, así como éstas anidadas dentro de otras.

Paralelamente se realizó la implementación de la arquitectura de palabras clave y el control de errores en la gestión de proyectos.

Realmente en esta primera etapa el resultado obtenido era lo más simple posible, sin la edición de estilos, el sistema interpretaba las etiquetas insertadas en el texto, y si eran correctas el sistema procesaba todos los archivos, además se usaba pandoc para procesar los archivos markdown y traducirlos a html y latex.

Se podría decir que el objetivo general de la iteración 1 fue la implementación de la estructura básica de HIDRA, tocando ligeramente algunas funcionalidades de la gestión de proyectos, y las salidas mínimas requeridas.

Iteración 2

Una vez implementada la estructura básica del sistema, en esta segunda iteración fueron realizándose distintas tareas de forma paralela. Por un lado la generación del formato ePub a través de html, lo que conllevó algunas tareas secundarias, como la de convertir las ecuaciones en imágenes de forma local para html, el procesado de las imágenes, comprimiéndolas y

con varias opciones de formato; tareas que queremos destacar por el esfuerzo que supuso.

Por otro lado, se trabajó en la redefinición de comandos latex en comandos propios, para facilitar la edición de los estilos por parte del usuario, la implementación de la bibliografía para el formato latex haciendo uso de la etiqueta de referencias.

Otros puntos importantes fueron la refactorización del sistema y la implementación de una *keyword* para la inserción de la bibliografía a partir de archivos estándar BibTex en html. Estas *keyword* son de vital importancia en el sistema, ya que se usan para tareas como la inserción de la tabla de contenidos, bibliografía o portada entre otras.

Además se desarrolló el primer ejemplo funcional del sistema. Muy sencillo, pero en el que podía observarse todas las funcionalidades de HIDRA.

Los avances principales de esta iteración pueden resumirse en el perfeccionamiento del formato de salida *ePub* y la inclusión de palabras clave.

Iteración 3

La tercera iteración se compuso de un gran conjunto de subtareas, de las cuales algunas estaban destinadas a la mejora de la estética de los documentos procesados y otras a la corrección de errores de código y su refactorización.

Algunas de estar tareas fueron la corrección del anti alising en las ecuaciones, las referencias en las fórmulas, la creación de subdirectorios para las distintas salidas, el procesado de las imágenes adecuándose a las salidas especificadas por el usuario. Además se implementó la etiqueta de cuadro de texto, y la inserción de la portada especificada por el usuario.

Se realizaron también distintas correcciones, que con el avance del proyecto iban surgiendo, como una serie de errores en ePub relacionados con la interpretación de las hojas de estilos css, las referencias de html y en consecuencia las de ePub (ya que el formato ePub se obtiene procesando la salida html), errores con las etiquetas de las tablas, imágenes y código, el posicionamiento de los enlaces a secciones e imágenes también tuvo que ser corregido, así como los cuadros de texto en el formato de libro digital, que se salían de los márgenes de la página y las referencias bibliográficas, que se cortaban y salían incompletas.

En resumen, podría decirse que los esfuerzos de esta iteración se destinaron a la corrección de errores.

Iteración 4

En esta iteración se introduce el concepto de SCORM en el sistema. Hasta el momento, los esfuerzos dirigidos a esta tecnología fueron puramente teóricos, si bien es cierto que se continúa con la investigación en este campo ya se comienza con la implementación del procesamiento de SCORM.

Tras el estudio de SCORM se definieron los elementos necesarios para su diseño, en con-

creto, en esta iteración se diseña la plantilla del *manifest*, que es un documento donde se encuentra reflejado el contenido y el orden que debe tener el paquete SCORM; se diseña también la división de la salida HTML en tantos archivos HTML autónomos como archivos de entrada definió el usuario, necesario para la estructuración de los elementos de aprendizaje del paquete SCORM.

Por otro lado se realizaron tareas de control de errores en argumentos de etiquetas, se creo un archivo *.log* para el loggeo del procesamiento de proyectos, refactorización general de algunas partes del código, como la organización de módulos y adaptadores.

Iteración 5

Durante esta iteración se introduce otro formato de salida, *MOBI* y se comienza a preparar el sistema para permitir la portabilidad.

Respecto al formato *MOBI*, finalmente se decidió sustituir por *azw*, debido a la dificultad de edición que presentaba el primero.

Paralelamente se procedía al empaquetado de dependencias para la versión de Windows de herramientas como *bibtex2html*, encargada de la conversión de ecuaciones a imágenes; *calibre*, para el procesamiento de los formatos de libros digitales; *imagemagick*, para el procesamiento de imágenes; *latex*, *pandoc* y *unzip*, las dos últimas, responsable de la conversión de archivos markdown a html y latex, y de la compresión de archivos respectivamente.

Otra tarea a destacar en esta iteración es el diseño inicial de la interfaz gráfica, hasta el momento todo se había realizado por línea de comandos.

Se realizaron algunas tareas independientes de las ya mencionadas, como comprobar si se va a generar HTML, en caso contrario no generar imágenes rasterizadas de las vectoriales, la refactorizar process y preprocess (clases que derivaron del HidraCore), la actualización de las dependencias del proyecto para Windows y Linux y la ejecución en hilo paralelo para no bloquear la interfaz del usuario al realizar el procesamiento.

Iteración 6

Esta iteración básicamente consistió en la implementación de un procesador de lenguaje propio para HIDRA. Tras el desarrollo de la misma se llegó a la conclusión que un correcto diseño que hubiera añadido el procesador inicialmente hubiera sido lo más deseable.

Pero a pesar de añadirlo en un momento tan tardío del desarrollo, las ventajas que ha proporcionado al proyecto han sido excepcionales, incorporando libertad en la inserción de etiquetas y facilitando el procesamiento de las mismas.

En el Anexo W puede verse el código que compone el analizador léxico y el analizador sintáctico.

Iteración 7

En la séptima iteración se realizaron algunos cambios y mejoras de la interfaz gráfica, como la implementación de la parte dedicada a la generación de los cinco formatos de salida disponibles hasta el momento: latex, html, epub, mobi, scorm; o reunir todas las ventanas en una sola utilizando "paneles" para evitar cambios de posición de la ventana.

También se inició una parte muy importante del proyecto: los test. Se comenzó con el estudio de la especificación de SCORM para test, realizando test a mano, para posteriormente implementar el procesamiento y la automatización de los mismos.

Se corrigieron algunos errores producidos por las descripciones de las etiquetas y el parser, y se implementaron otras nuevas, como las imágenes laterales.

Además se realizaron otras tareas de reorganización de directorios y archivos, y limpieza del código fuente, que junto a la refactorización de código hacen que el código se mantenga lo mas mantenible posible. La refactorización del código en tareas (*task*), que también se realizó en esta iteración, supuso un gran avance a la hora de incorporar nuevas tareas.

Por último se añadió un estilo denominado "Amazon", asegurándose que dicho formato pasaba la validación 6"x9" necesaria para la publicación de libros.

Iteración 8

En la última iteración se incorporaron una serie de mejoras tras las cuales se consideró que el sistema estaba finalizado.

Algunos de los cambios que se introdujeron fueron la posibilidad de añadir capítulos sin numeración en la tabla de contenidos, la mejora de salida de errores por pantalla para facilitar el uso de la aplicación al usuario, se arregló el estilo de las etiquetas *low level* y se permitió la inserción de imágenes en los test, que se implementaron para hacerlos multiformato (tanto SCORM, como pdf, html, azw3 y ePub).

Se realizaron ciertas modificaciones en el procesador de lenguajes, se añadió también otras nuevas etiquetas: las **ecuaciones en línea** y las **letras capitales**.

Respecto a los estilos, se modificó la plantilla *book* para que pasara la validación 6"x9" de Amazon, y se incluyó el estilo *TFG* basado en el establecido por la Escuela Superior de Informática.

Y por último se creó un asistente de instalación para Windows, en el Anexo I: Manual de Instalación Windows se describe el proceso de instalación.

Llegados a este punto, el proyecto se consideró finalizado y se procedió a la generación de la documentación.

5.2 Recursos y costes

En esta sección se describen los recursos, tanto temporales como económicos, empleados en el desarrollo de la plataforma, así como los costes asociados a la misma. También se incluye un apartado con las estadísticas asociadas al proyecto, obtenidas del repositorio, así como un análisis de rendimiento del software. Por último se detalla un ejemplo real de uso del sistema HIDRA.

5.2.1 Costes

El periodo de implementación de HIDRA comprende desde el 15 de junio del 2015 al 29 de julio del 2016. El trabajo de desarrollo no comprende íntegramente todo el periodo entre las fechas mencionadas, ya que el trabajo de implementación se desarrolló en el Centro de Tecnologías y Contenidos Digitales, y no se realizó únicamente este proyecto, la estimación de trabajo que se ha realizado es de 35 semanas, con una estimación de 6 horas diarias, lo que haría un total de 1050 horas de implementación aproximadamente. Considerando además un precio de unos 30 euros brutos por hora[1], el salario de los desarrolladores software ascendería a 31.500 cada uno, a los que se les añadiría el coste de los recursos empleados, que vienen desglosados en la siguiente tabla, y harían un total de 111.064,44 euros.

De la mayoría de los dispositivos usamos varias unidades para poder trabajar de forma paralela sin retención de recursos. El desarrollo software* es la parte correspondiente a uno de los desarrolladores, cuyo trabajo se realizó en 10 semanas.

Recursos	**Coste**	**Cantidad**	**Total**
Desarrollo software	31.500	3	94.500
Desarrollo software*	9.000	1	9.000
ASUS A55A	650	1	650
Workstation HP Z230	1.395	3	4.185
Pantalla HP Z27n	724,79	3	2.174,37
Bq - E-reader cervantes	147,72	2	295,44
Kindle Paperwhite	129,99	2	259,98

Cuadro 5.1: Coste asociado al desarrollo de la aplicación

5.2.2 Estadísticas

Para el desarrollo del sistema se ha utilizado un repositorio *Bitbucket*, que ha facilitado desde el primer momento la implementación del sistema de forma colectiva por todo el equipo de desarrollo, y el control de versiones.

[1]Estimación obtenido de www.infojobs.net

Empleando la herramienta *Gitstats* se ha obtenido un estudio del repositorio, que se muestra a continuación.

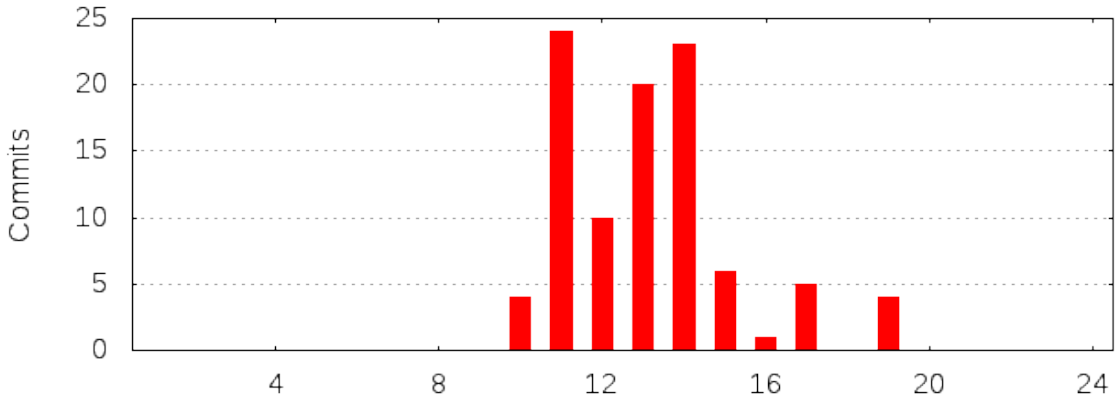


Figura 5.1: Actividad en el repositorio por horas del día

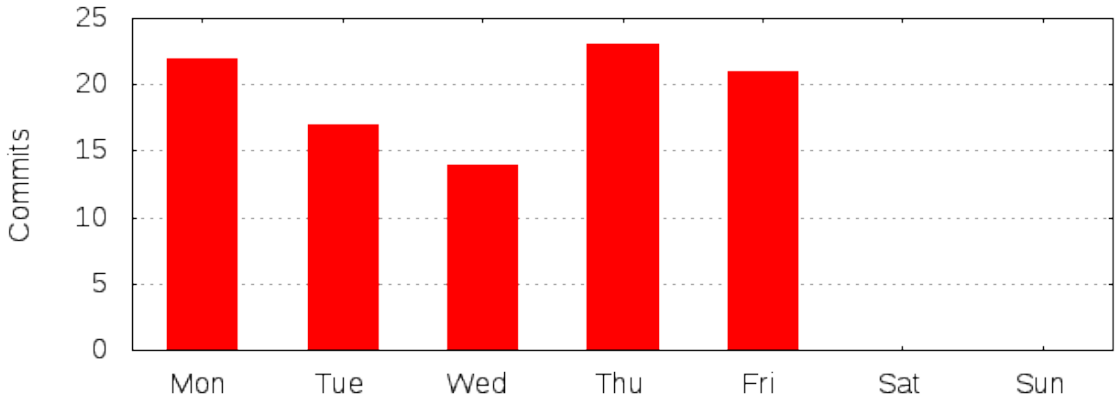


Figura 5.2: Actividad media en el repositorio semanalmente

Otros datos obtenidos por medio de *Gitstats* son los que se muestran en la tabla Lineas de código de los archivos según su extensión, de la que se han excluido los archivos que son autogenerados, o imágenes.

Extensión	**Archivos (%)**	**Líneas (%)**	**Líneas/archivo**
cls	4 (1.07 %)	1589 (9.55 %)	397
css	4 (1.07 %)	511 (3.07 %)	127
glade	1 (0.27 %)	493 (2.96 %)	493
hip	2 (0.53 %)	56 (0.34 %)	28
his	64 (17.07 %)	536 (3.22 %)	8
hit	122 (32.53 %)	677 (4.07 %)	5
html	3 (0.80 %)	636 (3.82 %)	212
js	4 (1.07 %)	1953 (11.74 %)	488

Extensión	Archivos (%)	Líneas (%)	Líneas/archivo
md	7 (1.87 %)	317 (1.91 %)	45
py	70 (18.67 %)	5047 (30.34 %)	72
sh	1 (0.27 %)	17 (0.10 %)	17
sty	15 (4.00 %)	1551 (9.32 %)	103
xml	3 (0.80 %)	26 (0.16 %)	8
Total	300 (77,61 %)	13409 (80,6 %)	-

Cuadro 5.2: Lineas de código de los archivos según su extensión

Por otro lado, se ha hecho uso de la aplicación *cloc* para contabilizar las líneas de código del sistema, lo que supone que las líneas de código autogeneradas también entrarán en el recuento. De las líneas totales que contabiliza, 549.565, son 13.409 las programadas, como se ha mencionado en el párrafo anterior.

Languaje	Archivos	En blanco	Comentarios	Código
Teamcenter def	526	459	253	230523
Lua	126	5778	7035	216975
Javascript	213	6622	11802	32786
HTML	130	4969	2139	24118
Visual Basic	36	603	0	22652
Perl	12	1456	5212	012
XML	21	105	296	3970
Python	71	1015	1063	3576
IDL	20	25	0	3575
CSS	19	390	27	2583
XSLT	3	95	73	975
Bourne Shell	34	190	434	847
XSD	4	127	76	844
SQL	1	0	0	549
DOS Batch	38	137	182	431
D	7	4	0	367
C/C++ Header	1	163	290	323
make	5	46	3	168
DTD	1	31	103	99
JavaServer Faces	2	5	0	93
awk	1	13	71	67

Languaje	Archivos	En blanco	Comentarios	Código
Java	1	2	14	32
Total:	1272	22235	29473	549565

Cuadro 5.3: Lineas de código de HIDRA contabilizadas con la aplicación cloc

5.2.3 Ejemplo práctico

Se ha decidido usar el presente proyecto para ilustrar el funcionamiento de HIDRA.

El primer paso fue la creación del proyecto, en el Anexo B: Crear un nuevo proyecto, se muestra el proceso para crear un proyecto nuevo, una vez que se ha generado el árbol de directorios, se procede a editar el archivo *Proyecto.hip*, que será editado cada vez que se deseen añadir nuevos archivos. En un momento concreto del desarrollo del presente documento, el archivo quedó como se muestra en el siguiente código:

```
general:
  images: imgs
  output: out
  documents: docs
  workdir: temp
  toclevel: 2
  style: tfg
  pdfsize: a4

project:
  abstract: abstract.md
  version: 1.0
  author: "Maria Alvarez Rodriguez"
  title: "Plataforma para la Edicion Liquida Multiformato"
  cover: Portada.jpg

documents:
  list:
  - toc
  - inicio.md
  - introduccion.md
  - objetivos.md
  - antecedentes.md
  - metodo.md
  - resultados.md
  - arquitectura.md
  - conclusiones.md
  - final.md
```

```
- anexoA.md
- anexoB.md
- anexoC.md
- anexoD.md
- anexoE.md
- anexoF.md
- anexoG.md
- anexoH.md
- anexoI.md
- bibliography
```

Código 5.1: Contenido del archivo Proyecto.hip

En este caso, seleccionamos la opción de estilo **tfg**, pero para ilustrar lo sencillo que es cambiar con HIDRA el estilo, sin tocar el contenido del documento, se adjunta la presente documentación con un estilo diferente al requerido por la universidad.

Por otro lado, en la imágen se muestra el árbol de directorio, donde puede apreciarse que los documentos se encuentran en la carpeta *docs*, las imágenes a las que se hace referencia, además de las proporcionadas por HIDRA, como son las de los cuadros de texto, se encuentran en la carpeta *imgs*, en la carpeta *style* aparecerán las hojas de estilos correspondientes al seleccionado en el fichero *Proyecto.hip*, y las carpetas *temp* y *out* incluirán los archivos temporales y finales generados por el sistema, respectivamente.

La edición de los documentos y la incorporación de las imágenes es tarea del usuario, sin embargo, el estilo viene completamente definido, permitiendo la edición de los elementos que se desee.

Por otro lado, no es necesario generar el proyecto nuevamente cada vez que se inicia la aplicación, está la opción **Abrir proyecto**, que viene detallada en el Anexo C: Abrir un proyecto, que permite seleccionar la ruta del proyecto, y escoger los formatos de salida que se deseen.

Al tener que recompilar el proyecto para incluir nuevos cambios, es recomendable hacerlo cada poco tiempo.

Para ilustrar el resultado de procesar un proyecto y obtener distintos formatos, el día de la presentación se mostrarán todas las salidas, en los lectores digitales correspondientes y en versiones impresas, aunque en las siguientes imágenes se muestran algunas capturas en emuladores de libros digitales, moodle e incluso fotografías de los dispositivos.

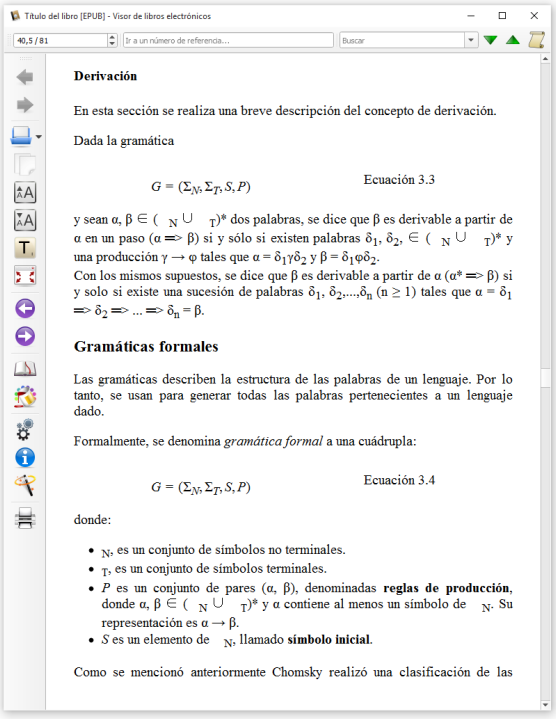


Figura 5.3: Captura de un emulador de lector de libros electrónicos

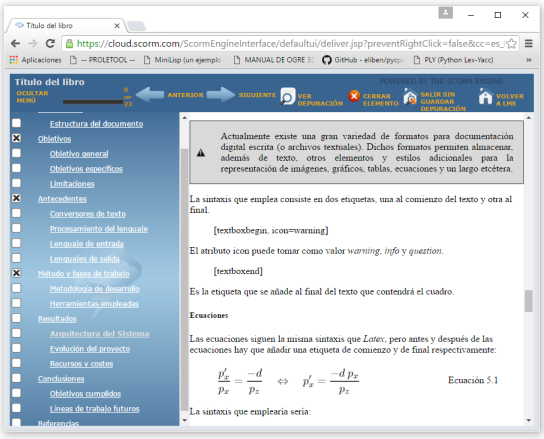


Figura 5.4: Captura de un reproductor de SCORM

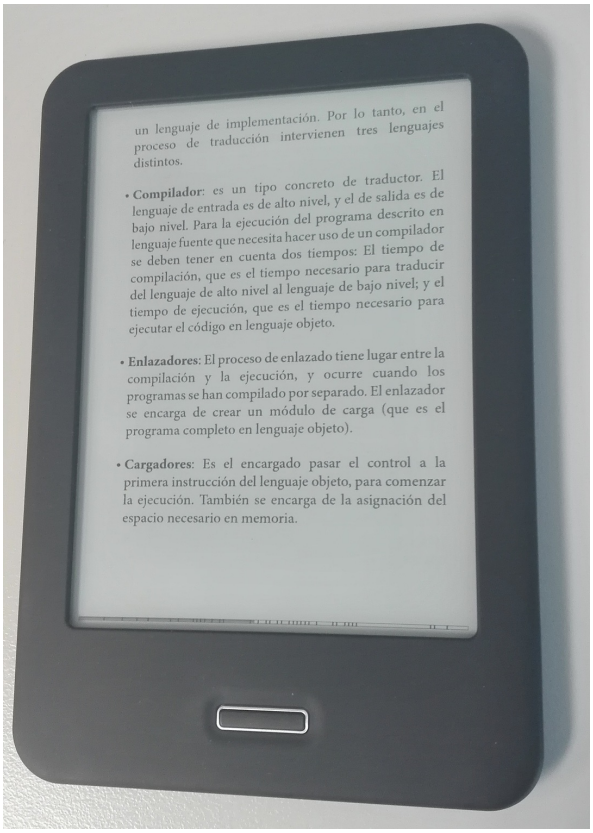


Figura 5.5: Lector de libros electrónicos en formato ePuB

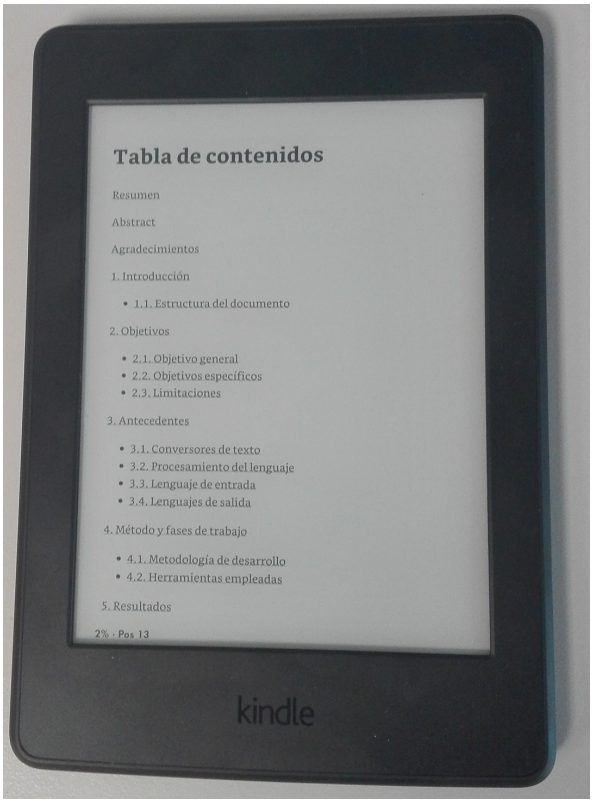


Figura 5.6: Lector de libros electrónicos en formato azw

Capítulo 6

Arquitectura del Sistema

En este capítulo se describe la arquitectura de HIDRA, usando un enfoque *Top-Down*, en el que se realiza una descripción general del sistema, y se continuará con definiciones cada vez más detalladas de los módulos que componen el sistema. El sistema HIDRA está formado por los siguientes módulos, que pueden apreciarse en la figura:

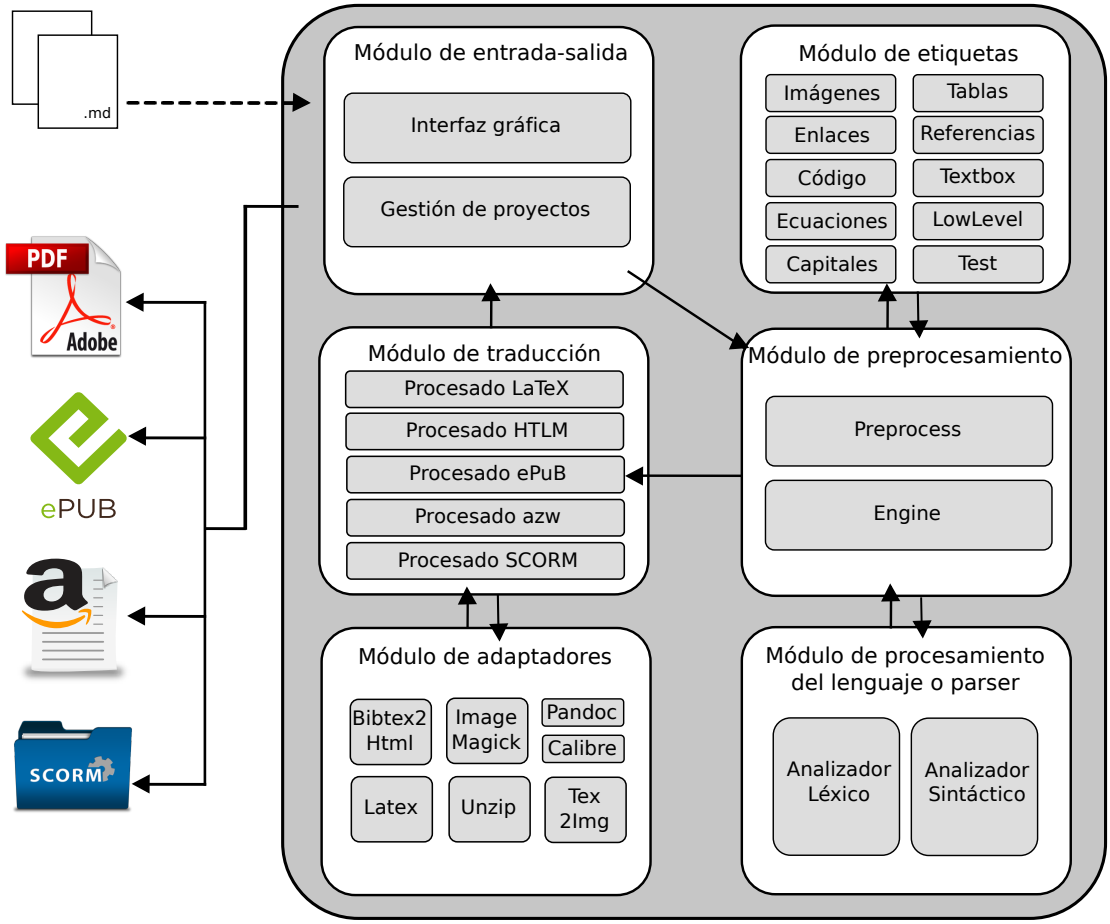


Figura 6.1: Esquema de la estructura modular de HIDRA

- ***Módulo de etiquetas***: Define las estructuras que dan soporte a los distintos elementos

de maquetación.

- ***Módulo de entrada-salida:*** Es el encargado de realizar la comunicación del sistema con el exterior.
- ***Módulo de preprocesamiento:*** Obtiene las estructuras que el procesador de lenguajes se encargará de transformar.
- ***Módulo parser o de procesamiento del lenguaje:*** Realiza un análisis léxico, sintáctico y semántico para generar el código correspondiente a las estructuras obtenidas por el *Módulo de preprocesamiento*.
- ***Módulo de traducción:*** Genera el formato en las distintas salidas especificadas.
- ***Módulo de adaptadores:*** Módulo que se encarga de la adaptación de las herramientas externas que se usan en el sistema.

En la imágen 6.2 puede observarse el diagrama de clases simplificado, para ilustrar más cómodamente el funcionamiento del sistema, dónde *process* representa los cinco posibles procesamientos que se desarrollarán en el *módulo de traducción*.

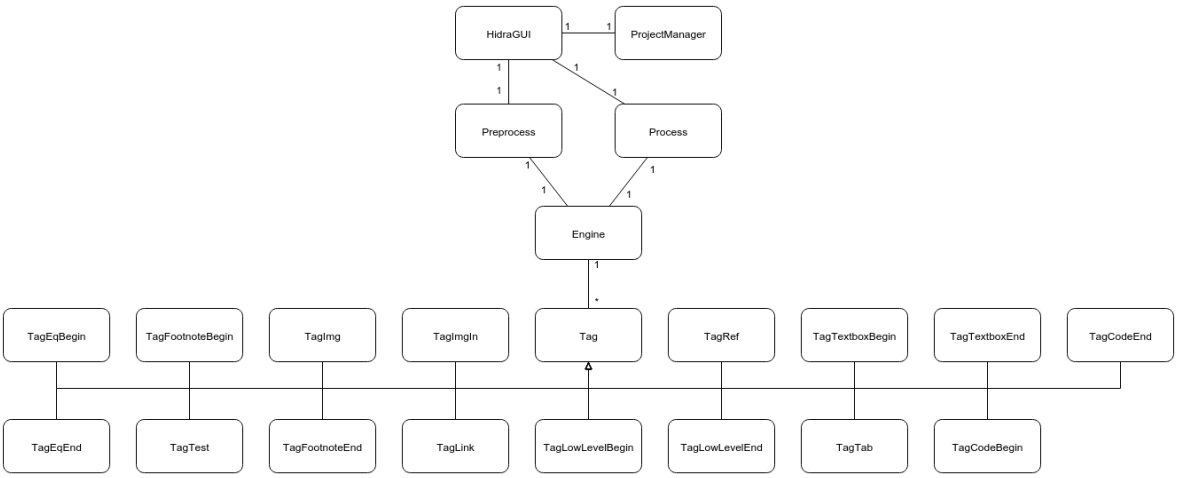


Figura 6.2: Diagrama de clases

Por otro lado, la imágen 6.3, muestra un diagrama de secuencia del sistema, donde puede verse, de forma generalizada, el funcionamiento del mismo.

6.1 Módulo de etiquetas

Para dar soporte a las distintas necesidades que iban teniendo lugar a la hora de maquetar documentos se han definido distintos tipos de etiquetas, y son la base del sistema, las que permiten que sea multiplataforma, la inserción de las etiquetas en la documentación es muy sencilla, en algunos basta con incluir una línea código en el lugar de por ejemplo, una imagen, o en el caso de las tablas, se inserta la tabla en formato *Markdown* y luego una línea de código. Puede parecer algo enrevesado al explicarlo de esta forma, pero un ejemplo saca de dudas.

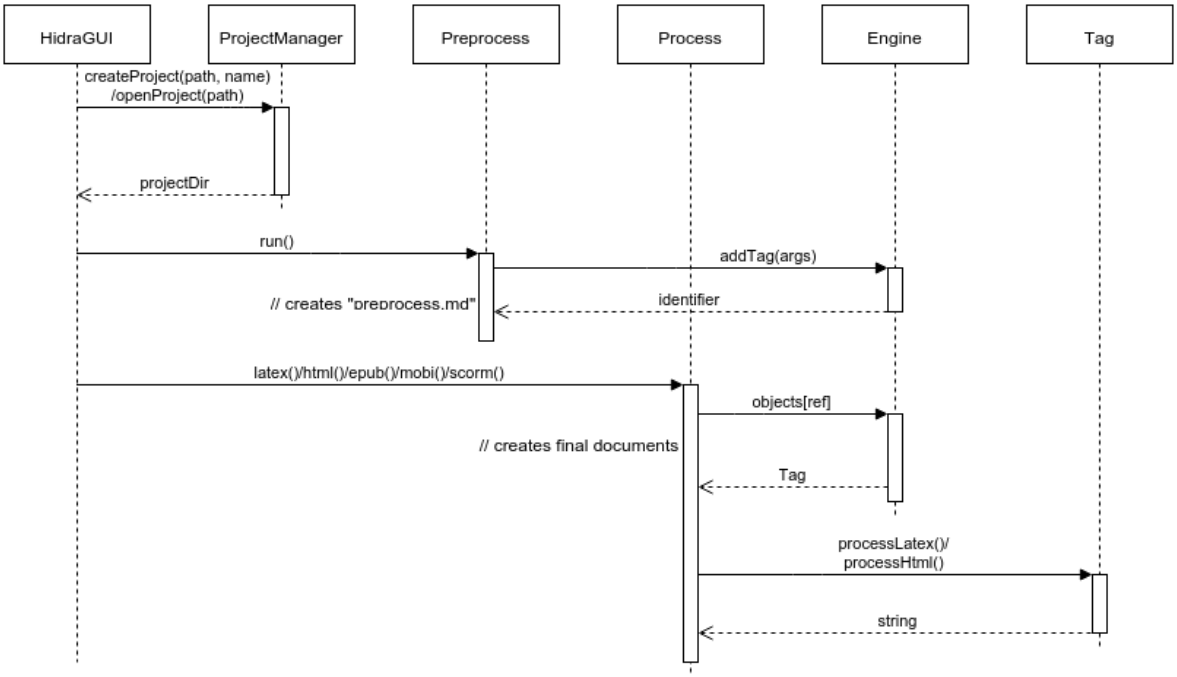


Figura 6.3: Diagrama de secuencia

Con HIDRA es muy sencillo obtener un mismo documento en distintos formatos, pero no simplemente traduciendo lenguajes, como hacen algunas herramientas, si no que adapta el contenido a cada tipo de formato, la inserción de una tabla en un documento es distinta según el formato que se desee, por ejemplo, la siguiente tabla:

Markdown	Less	Pretty
Still	`renders`	**nicely**
1	2	3

Cuadro 6.1: Tabla de ejemplo

Si se quisiera insertar en un archivo *.lex*, sería de la siguiente forma:

```
\begin{longtable}[c]{@{}lll@{}}
\toprule
Markdown & Less & Pretty\tabularnewline
\midrule
\endhead
\emph{Still} & \lstinline!renders! & \textbf{nicely}\tabularnewline
1 & 2 & 3\tabularnewline
\bottomrule
```

Código 6.1: Tabla de ejemplo latex

En cambio para *.html* sería, por ejemplo:

```
<td align="left">1</td>
<td align="left">2</td>
<td align="left">3</td>
</tr>
</tbody>
</table>
```

Código 6.2: Tabla de ejemplo html

A simple vista, puede apreciarse el esfuerzo que supondría ir desarrollando dos documentos paralelos en distintos formatos. HIDRA pone fin a este problema, gracias al módulo de etiquetas, una misma implementación sirve para todos los formatos. Esto es posible ya que en el preprocesamiento se detectan estas implementaciones comunes y se convierten en objetos de etiquetas, que más tarde se traducirán a los formatos correspondientes. En el Anexo D: Sintaxis de etiquetas propias, se describe y se muestran ejemplos de como añadir cada tipo de elemento.

El *módulo de etiquetas* se compone de tantas clases como elementos de maquetación se adapten en el sistema, actualmente son:

- Imágenes
  - Imágenes simples
  - Imágenes in line
  - Imágenes laterales
- Tablas
- Enlaces
- Referencias
- Código
- Cuadros de texto
- Ecuaciones
- Código de bajo nivel
- Letra capitales
- Test

En el *módulo de preprocesamiento* se describe qué y cómo se obtienen los atributos que se utilizan en los constructores de las etiquetas, ver para más precisión. El *módulo de etiquetas* cuenta con una clase principal, **Tag** de la que heredan las etiquetas más específicas. Sus atributos son un identificador único que se genera automáticamente en el constructor de la clase, el *projectManager* y una variable que indica si la etiqueta es de bloque o no; además

de unos métodos para obtener o modificar el contenido de las variables mencionadas, y los métodos más importantes: **processLatex** y **processHtml**.

Cada clase en su constructor contiene unos atributos que dependen del tipo de etiqueta que sean, por ejemplo, en el caso de las imágenes algunos de sus atributos son **filepath**, que almacena la ruta de la imagen, **percentaje** o **desc**, que son el ancho de la imágen y la descripción de la etiqueta respectivamente; en cambio, la de código tiene **lang**, que contiene el lenguaje del código; el constructor se encarga de dar el valor correcto de estos atributos, obtenidos del *módulo de preprocesamiento*, se muestra un ejemplo en el siguiente código:

```
def __init__(self, projectManager, args):
    super(self.__class__, self).__init__(projectManager)

    self.type = strings.Tags.TAGCODEBEGIN
    self.tags = []
    self.isBlock = True
    for i in xrange(0, len(args[1])):
        s = args[1][i]
        if s[0] == strings.References.LANGUAGE:
            self.lang = s[1]
        elif s[0] == strings.References.DESCRIPTION:
            self.desc = s[1][1:-1]
        elif s[0] == strings.References.REFERENCE:
            self.idRef = s[1]
        elif s[0] == 'ncode':
            self.ncode = s[1]
        else:
            if s[0][0] == strings.Tags.TAGREF:
                key_options = s[0][1][0]
                ref = tag_ref.TagRef(self.projectManager, key_options)
                self.tags.append(ref)
```

Código 6.3: Constructor TagCodeBegin

Una vez almacenados los atributos, y dependiendo del procesado que sea necesario se invocan los métodos **processLatex** y **processHtml**. Cada formato cuenta con una serie de plantillas para cada etiqueta, además de tener definidos los estilos para cada uno de ellos previamente, en estos métodos se lee el fichero que contiene la plantilla que corresponda, y sustituye las variables por el valor que tiene. Por ejemplo, continuando con el ejemplo de código, para latex, el contenido de la plantilla sería:

$begin\{code\}\{\$lang\}\{\$ncode : \$desc\}\{\$idref\}$

donde las variables se identifican por el símbolo que las precede. Y su sustitución en el método *processLatex*:

```
s = Template(file.read())
return s.substitute(idref=self.idRef, lang=self.lang,
                    ncode=self.ncode, desc=self.desc)
```

Código 6.4: Fragmento de código de processLatex

Para html el proceso es el mismo, y para el resto de formatos que no se han mencionado, como ePub, azw o SCORM, al derivar de html, es suficiente con la versión en *html*.

6.1.1 Gestión de imágenes

La gestión de las imágenes, es una tarea muy especializada en los distintos formatos de salida del sistema, ya que cada uno de ellos tiene necesidades diferentes, y con HIDRA quedan satisfechas, por ejemplo, para el formato html, haciendo uso de **ImageMagick** las imágenes son procesadas para obtener una versión más ligera de las mismas, en formato *.jpg*. Para latex, si es posible, el sistema mantendrá la versión vectorial, o en su defecto las procesará también.

De todas estas tareas se encarga la clase **TaskPrepareImages**, cuyo método de preprocesamiento puede observarse en el listado de código siguiente:

```
def _preprocessImage(self, formato):
    projectDir = self.projectManager.project.projectDir

    list = os.listdir(os.path.join(projectDir, 'imgs'))
    for l in list:
        image = l.split('.')
        Imagemagick().run(os.path.join(projectDir, 'imgs',
                                       image[0] + '.' + image[1]),
                          os.path.join(projectDir, 'imgs',
                                       image[0] + '.jpg'))
        Imagemagick().resize(os.path.join(projectDir, 'imgs',
                                          image[0] + '.jpg'),
                             os.path.join(projectDir, 'imgs',
                                          image[0] + '.jpg'), formato)
```

Código 6.5: Método preprocessImage

6.1.2 Gestión de ecuaciones

En el Anexo D: Sintaxis de etiquetas propias, puede observarse en más detalle la definición de las ecuaciones, pero básicamente, siguen la estructura de *latex*. De esta forma, el procesamiento *latex* no presenta dificultades, en cambio para *html* y sus derivados necesita una preparación específica.

Para ello, se hace uso de **Tex2Im**, que hace uso de latex para compilar la ecuación insertada, y así obtener una imágen de la misma, que será lo que se insertará en *html, ePuB, azw* y *SCORM*.

6.2 Módulo de entrada-salida

El *módulo de entrada-salida* está constituido principalmente por la interfaz gráfica, pero también la forma la estructura de ficheros de cada proyecto y los archivos que el usuario añade en él.

La interfaz gráfica está constituida por tres ventanas distintas. La primera es la que se ve en la imágen 6.4.

Figura 6.4: Pantalla de inicio

Es muy sencilla e intuitiva, tiene tres botones:

- Nuevo
- Abrir
- Salir

Desde la opción **Nuevo** se tiene acceso a la segunda ventana de la interfaz 6.5, que permite crear un proyecto.

Esta ventana cuenta con distintos elementos, un *cuadro de texto* para introducir el nombre del proyecto; una *lista desplegable*, para seleccionar la ruta; y tres botones. En el Anexo B: Crear un nuevo proyecto, se describe la creación de nuevos proyectos, ver para más detalle.

Figura 6.5: Nuevo proyecto

La tercera ventana, a cual se accede desde la opción **Abrir** de la primera ventana, o **Siguiente de la segunda**, presenta la siguiente apariencia:

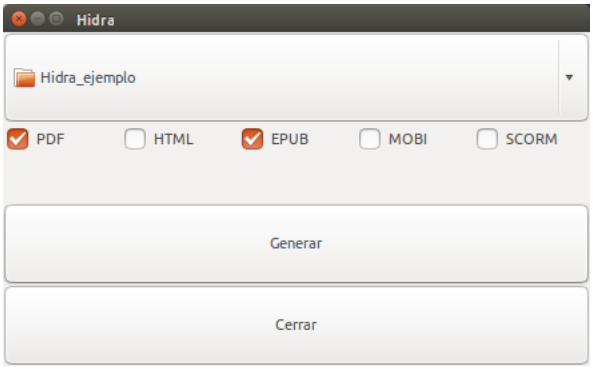


Figura 6.6: Generación de los formatos finales

Una *lista desplegable* para seleccionar el proyecto que se desea procesar; un *check list*, para seleccionar entre los posibles formatos de salida, y dos botones.

Además, todas las ventanas tienen una barra con dos opciones, **Archivo** y **Ayuda**, desde la opción de ayuda se accede a un manual de uso en versión html, o a una ventana que muestra información sobre la aplicación:

Esta interfaz ha sido desarrollada en *Glade*, usando gtk 2.24, el archivo *GUIHidra.glade* resultante puede verse en el DVD que se adjunta al presente documento.

6.3 Módulo de preprocesamiento

Otro de los principales módulos que componen el sistema es el de **preprocesamiento**. Los primeros pasos que da el sistema a la hora de generar la documentación resultante son la preparación de las imágenes cuando es necesario y la generación del archivo *temp/full.md*, este archivo consiste en la concatenación de todos los documentos creando uno único, además si encuentra una palabra clave, la inserta en el final como una etiqueta.

El módulo de preprocesamiento parte de este archivo y el archivo resultante es *temp/pre-*

Figura 6.7: Acerca de...

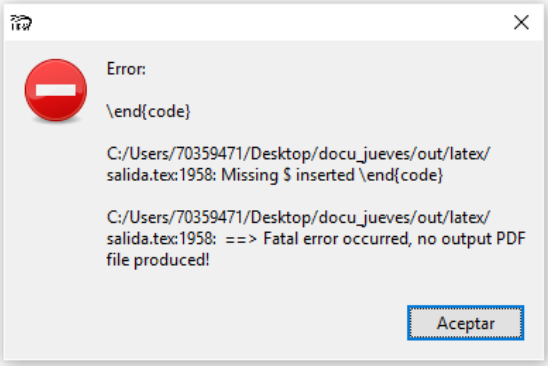


Figura 6.8: Ejemplo de salida errónea

procesado.md. El objetivo principal de este módulo es detectar los elementos que necesitan cierto procesamiento para ser introducidos en la documentación, ya sean imágenes, estilos de texto, tablas o código entre otros. En concreto, cuando se trata de las etiquetas definidas para HIDRA hay un flujo de información con el *módulo parser o de procesamiento del lenguaje*.

Este módulo se ha implementado básicamente en dos clases, por un lado la clase **TaskPreprocess** y la clase **Engine**. La primera, se encarga de detectar las estructuras que son posibles etiquetas, éstas se envían al *parser* que devolverá una estructura con los argumentos de la etiqueta si ha sido correctamente insertada. Con esta estructura el preprocesador se encargará de construir un objeto etiqueta, pero para ello se analiza, y dependiendo su tipo se aplican una serie de métodos, por ejemplo, si es una etiqueta anidada, entendiéndose como tal aquellas que constan de una etiqueta de apertura y otra de cierre, tendrá un tratamiento distinto si es una etiqueta de ecuación. Si la etiqueta es de imagen, ecuación, tabla o código se invoca a los métodos correspondientes que calculan su referencia dentro del documento, y se añade como atributo al array que proporcionó el *parser*; por otro lado si trata de una etiqueta de bloque, se almacena el contenido de la misma, que será necesario en el módulo de traducción.

TaskPreprocess hereda de la clase **HidraTask**, que puede verse en el anexo K, de dicha clase heredan todas las tareas del sistema, tiene como único atributo una instancia de *ProjectManager*, es una clase fundamental del sistema, instancia un objeto *Project*, que almacenará todas las características del mismo, véase titulo, autor, estilo, rutas de los directorios entre otros, y métodos para trabajar con ellos. En Anexo L: Código fuente de ProjectManager, y Anexo M: Código fuente de Project, se encuentra el código fuente de las clases *ProjectManager* y *Project*. Volviendo a la clase **TaskPreprocess**, consta de una serie de atributos que se irán modificando según el tipo de etiqueta que se vaya procesando, como el capítulo actual, el número de ecuación o si se está procesando un fragmente de código fuente, pero hay dos atributos más que destacar:

```
self.procesador = Hidra()
self.engine = engine
```

Código 6.6: Algunos atributos de TaskPreprocess

El primero de ellos crea una instancia del procesador de lenguajes, y por medio de dicho atributo es como se realiza la comunicación con el *módulo parser o de procesamiento del lenguaje*:

```
list= self.procesador.default(cad.group(0).encode('utf-8'))
```

Código 6.7: Instrucción para invocar al parser

donde, *cad* es la cadena que podría ser una etiqueta, y *default* el método que analiza la cadena de entrada por defecto. A su vez el *parser* devuelve un array si es una etiqueta bien construida, este array está divido por argumentos, *list*. Por ejemplo, si la etiqueta fuese:

> [img, path=proyecto-hip.png, width=100, label="Ejemplo de archivo de proyecto (Proyecto.hip)", ref=ProyectoHip]

El array sería **[img, [[path, proyecto-hip.png], [width, 100], [label, "Ejemplo de archivo de proyecto (Proyecto.hip)"], [ref, ProyectoHip]]]**

Con este array se procedería a invocar el método más importante de la clase **TaskPreprocess**, que se encarga de reunir los argumentos necesarios para invocar a la clase **Engine**, que es la encargada de construir el objeto etiqueta correspondiente según el tipo indicado. Los argumentos que se le proporcionan son:

- **Type**: corresponde al primer argumento del array proporcionado por el parser.
- **Key_options**: array contenida en el segundo argumento de *list*, consta de todos los argumentos de la etiqueta.
- **Tag_aux**: algunas etiquetas necesitan de otra para su correcto funcionamiento, como son las etiquetas de bloque.

La clase **Engine** recibe estos argumentos, y con ellos invoca al constructor correspondiente, o en algunos métodos especiales al método que invocará al constructor tras un pequeños cambios. Para ello, consta de un diccionario, cada etiqueta, salvo las que cierran un bloque, se relaciona con el constructor de la clase correspondiente.

```
self.tags = {
            strings.Tags.TAGCAP: tags.tag_cap.TagCap,
            strings.Tags.TAGIMG: tags.tag_img.TagImg,
            strings.Tags.TAGIMGIN: tags.tag_img_in.TagImgIn,
            strings.Tags.TAGIMGLAT: tags.tag_img_lat.TagImgLat,
            strings.Tags.TAGTAB: tags.tag_tab.TagTab,
            strings.Tags.TAGREF: tags.tag_ref.TagRef,
            strings.Tags.TAGLINK: tags.tag_link.TagLink,
            strings.Tags.TAGCODEBEGIN: tags.tag_code_begin.TagCodeBegin,
            strings.Tags.TAGCODEEND: self._endCode,
            strings.Tags.TAGEQBEGIN: tags.tag_eq_begin.TagEqBegin,
            strings.Tags.TAGEQEND: self._endEq,
```

```
        strings.Tags.TAGEQINBEGIN: tags.tag_eq_in_begin.TagEqInBegin,
        strings.Tags.TAGEQINEND: self._endEqIn,
        strings.Tags.TAGLOWLEVELBEGIN: tags.tag_low_level_begin.TagLowLevelBegin,
        strings.Tags.TAGLOWLEVELEND: self._makeLowLevel,
        strings.Tags.TAGTEXTBOXBEGIN: tags.tag_textbox_begin.TagTextboxBegin,
        strings.Tags.TAGTEXTBOXEND: tags.tag_textbox_end.TagTextboxEnd,
        strings.Tags.TAGFOOTNOTEBEGIN: tags.tag_footnote_begin.TagFootnoteBegin,
        strings.Tags.TAGFOOTNOTEEND: tags.tag_footnote_end.TagFootnoteEnd
    }
```

Código 6.8: Diccionario de etiquetas de Engine

Los objetos "tags" que son construidos se almacenan en un diccionario, donde se ordenan por una clave que proporciona cada uno de los constructores, y es único. Este identificador es el que se inserta en el documento *temp/preprocesado.md*, es el *módulo de traducción* quien se encargá de las etiquetas a partir de este momento.

6.4 Módulo parser o de procesamiento del lenguaje

Crear un lenguaje de etiquetas propio planteaba muchas ventajas, pero también inconvenientes. Al comienzo del desarrollo se decidió usar expresiones regulares para el tratamiento de las etiquetas de HIDRA. El principal problema de esta decisión es que no es una solución escalable. Cuando se incluían nuevas etiquetas con estructuras diferentes, había que modificar muchas partes en el código, además cuando las etiquetas tenían estructuras muy complejas las expresiones regulares se complicaban demasiado, y no abarcaban todos los casos necesarios.

Algunas de las expresiones que se llegaron a usar pueden verse en el listado de código 6.9, donde se aprecia la complejidad de las mismas.

```
\[ \w+ ((\s*,\s*)((\w+=(.+|/?(\w+/)*\w+))))* \]
\[\w+((,\s\w+=(/*\w\s*\.*:*/*)+(/\w+)*)*(,\s\[\w+(,\s\w+=(/*\w\s*)+(/\w+)*)*(,\s)*\])*)*\]
```

Código 6.9: Ejemplo de expresiones regulares de etiquetas

Debido a estos problemas se decidió usar un procesador del lenguaje. En el capítulo Estructura de un compilador, que consta de dos fases: la **fase de análisis** y la **fase de síntesis**, a su vez, la fase de análisis se divide en *análisis léxico, sintáctico* y *semántico*. Debido a la sencillez del análisis se ha optado por implementar un procesador de lenguaje que se encargue del análisis léxico y el sintáctico.

Para la implementación de dicho procesador o *parser* se ha hecho uso de una herramienta, mencionada también con anterioridad, para más detalles ver Python-ply, es una herramienta

que a partir de un vocabulario y una gramática genera un analizador. En este caso el analizador es **LALR**, por lo que se realiza un análisis ascendente.

Para ello, se implementaron dos archivos, *léxico.py* y *yacc.py*. En el primero, se definen los **tokens** que constituyen el vocabulario, como puede verse en el código 6.10:

```
tokens = (
                'LCORCHETE',
                'RCORCHETE',
                'COMA',
                'IGUAL',
                'IDENTIFICADOR',
                'SIMBOLOS',
                'LITERAL'
        )
```

Código 6.10: Definición de tokens

Y se implementan las expresiones regulares que definen cada uno de esos tokens. Además pueden definirse funciones de error, para determinar qué mensaje mostrar y con qué datos en caso de error léxico. El módulo **lex** de *Python-ply* se encarga de construir el analizador léxico con los elementos que acaban de ser descritos.

Por otro lado, se encuentra *yacc.py*, que está compuesto por una serie de funciones que representan las producciones de la gramática, además pueden definirse funciones adicionales que ayuden a simplificar las producciones, en el código 6.11, se muestra la función que define la primera producción de la gramática.

```
def p_tag(p):
    'tag : LCORCHETE IDENTIFICADOR lista RCORCHETE'
    p[0] = [p[2],p[3]]
```

Código 6.11: Definición de producción

Al igual que *lexico.py* también puede definirse una función de error. Es **yacc** quien se encarga de importar el léxico anteriormente definido, y a su vez, en el preprocesamiento de los archivos de proyecto se invoca a yacc para el análisis de flujo de datos obtenido de los mismos.

A pesar de que *python* sea un lenguaje interpretado, python-ply *compila* estas clases para obtener una tabla de símbolos y una máquina de estados.

En el Anexo J: Salida generada por el parser, se observa el archivo *parser.out* completo, que muestra, en primer lugar, una lista con las reglas de la gramática, pueden verse en el siguiente fragmento de código 6.12:

```
Rule 0     S' -> tag
Rule 1     tag -> LCORCHETE IDENTIFICADOR lista RCORCHETE
Rule 2     atributo -> elemento lista
Rule 3     atributo -> tag lista
Rule 4     lista -> COMA atributo
Rule 5     lista -> empty
Rule 6     empty -> <empty>
Rule 7     elemento -> IDENTIFICADOR IGUAL opciones
Rule 8     opciones -> cadena
Rule 9     opciones -> LITERAL
Rule 10    cadena -> IDENTIFICADOR cadena
Rule 11    cadena -> SIMBOLOS cadena
Rule 12    cadena -> empty
```

Código 6.12: Reglas de la gramática

A continuación, los terminales y los no terminales que componen la gramática, y las reglas en las que aparecen. Y por último, una máquina de estados, que en concreto consta de 22 estados diferentes.

6.5 Módulo de traducción

Hasta ahora, los módulos se han encargado de la gestión del proyecto, el diseño de etiquetas o el preprocesamiento de las mismas, pero aún no se ha descrito ninguna de las funcionalidades que permiten obtener los formatos de salida. Esto es tarea del *módulo de traducción*. Se podría decir que se solapa en ocasiones con el *módulo de etiquetas*.

El punto inicial del que parte este módulo es el archivo *temp/preprocesado.md* generado del preprocesamiento, indicar que este archivo contiene la información del proyecto en texto plano, y en lugar de los elementos de las etiquetas, aparece un identificador de la forma:

```
HIDRA:img:d452d16f-4c0844d5-a719-534c4b8bab0e
```

Código 6.13: Ejemplo de identificador de etiqueta

Además, las marcas de *Markdown* se mantienen. A partir de este punto la tarea se divide, según el formato al que se quiera convertir el proyecto. Para el caso de LaTeX, hay un paso adicional, además del procesamiento hay que realizar una compilación, para obtener el

formato final en PDF; y para ePuB, azw y SCORM, es necesario realizar antes el procesado html.

6.5.1 Procesado LaTeX

Partiendo del archivo *temp/preprocesado.md*, el primer paso es convertir el archivo a lo *bruto*, usando **Pandoc**, en el listado de código se observa la instrucción que usa Pandoc para traducir el archivo preprocesado.md a preprocesado.tex.

```
Pandoc().run(os.path.join(project.workDir(), 'preprocesado.md'),
             ['--listing', '--webtex', '--chapters'],
             os.path.join(project.workDir(), 'preprocesado.tex'))
```

Código 6.14: Conversión a .tex

El resultado es un archivo *.tex* en el que las marcas de *Markdown* se han transformado al formato latex. A continuación hay que insertar las etiquetas almacenadas por el *módulo de preprocesamiento*. Para ello, se procede a leer el archivo que se acaba de obtener hasta encontrar los identificadores de etiqueta, para cada una de las cuales se obtiene el objeto que se almacenó en el diccionario; luego, se usa la función **processLatex** que tienen implementada cada uno de estos objetos, y cuyo retorno es una cadena que se inserta en el documento sustituyendo al identificador de etiqueta. En algunos casos, además es necesario realizar ciertos arreglos, ya que *Pandoc* puede introducir caracteres sobrantes o elementos que no interesan.

En el archivo *hidra/tasks/task_process_latex.py.py* en el DVD adjunto puede verse el proceso completo.

El resultado es el archivo *temp/procesado.txt*, del que se realizará una copia en el directorio *latex* de los formatos finales, con el nombre de *salida.tex*. Este es el archivo que se procederá a compilar, dos veces, para que las referencias del texto se muestren correctamente.

6.5.2 Procesado Html

El procesado en html sigue un proceso muy parecido al procesado latex, la instrucción que usa Pandoc para convertir *preprocesado.md* a *preprocesado.html* se muestra en el siguiente fragmento de código.

```
Pandoc().run(os.path.join(project.workDir(), 'preprocesado.md'),
             ['-s', '-S', '--highlight-style=pygments'],
             os.path.join(project.workDir(), 'preprocesado.html'))
```

Código 6.15: Conversión a .html

Como en el subapartado anterior, el siguiente paso sería insertar las etiquetas almacenadas por el *módulo de preprocesamiento* en este caso se hace uso de la función **processHtml**, y también son necesarias algunas modificaciones. En este caso, el archivo donde puede verse en detalle el procesamiento latex es *task_process_html.py*.

6.5.3 Procesado ePuB

Para el procesamiento ePub se usa como base el archivo resultante del procesado html, pero el proceso que sigue se complica. Al igual que en los casos anteriores, se hace uso de **Pandoc** para una primera conversión, y además se hace uso de **Unzip** para las descompresión del resultado de la conversión, ya que hay que editarlo.

```
Pandoc().run(os.path.join(project.workDir(),
                          'salida_no_cover.html'),
             ['--highlight-style=pygments', '--webtex',
              '--epub-cover-image=' + os.path.join('images', project.cover())],
             os.path.join(project.workDir(), 'temp.epub'))
Unzip().run(os.path.join(project.workDir(), 'temp.epub'), project.tempEpub())
```

Código 6.16: Conversión a .epub

A continuación hay que realizar distintas tareas, por un lado, concatenar todos los estilos en un único CSS, en segundo lugar, obtener la lista de XHTMLs generados por Pandoc y añadirles los estilos. Por otro lado las imágenes deben añadirse al archivo epub, y para ello hay que prepararlas, por medio de **Imagemagick**:

```
Imagemagick().resizeForEpub(os.path.join(project.tempEpub(), strings.Project.HTML_IMAGES, img))
```

Código 6.17: Preparación de imágenes para ePub

El siguiente paso sería añadir las rutas de las imágenes al "content.opf", autor y titulo.

Y por último comprimir el ePub ya procesado. Para ello se copian los archivos generados, los media y META-INF.

6.5.4 Procesado Azw

Para el procesado MOBI o azw, que es una variante del primero, se hace uso de **Calibre** y simplemente se convierte, sin realizar cambios posteriormente.

```
Calibre().run(os.path.join(project.workDir(), 'temp.epub'),
                    os.path.join(project.outputDirMobi(), 'salida.azw3'),
```

```
os.path.join(project.tempEpub(), 'content.opf'))
```

Código 6.18: Conversión a .azw

6.5.5 Procesado SCORM

Para comenzar el procesado SCORM, lo primero es separar *salida.html*, que es el resultado del procesado html, en varios archivos, tantos como capítulos tenga el libro, o secciones en las que se divida el proyecto. Cada uno de estos capítulos necesita algunas modificaciones, como expresiones sobrantes o caracteres que hay que cambiar, pero la dificultad en este procesado consiste en la edición del *imsmanifest.xml*.

La estructura que debe tener este documento es muy concreta, y distinta en cada proyecto, la idea se basa en la división inicial del archivo *salida.html*, dependiendo de su posición en el proyecto, a los distintos capítulos se le asigna un nombre con una estructura determinada:

file-capitulo_subcapitulo.html

donde capítulo y subcapítulo se sustituirán por los números correspondientes. A la hora de generar el archivo *imsmanifest.xml* se basa en estos nombres, listando los archivos y añadiéndolos donde corresponde. Además, es necesario conocer los archivos media que contiene cada capítulo, para cambiar las rutas y añadirlos también.

El proceso completo puede verse en el Anexo O: Procesamiento SCORM.

6.5.6 Submódulo de keywords

Se ha mencionado a lo largo del presente documento, pero las *keywords* son imprescindibles en el proyecto para algunas tareas como la inserción de una portada, la tabla de contenido o la bibliografía.

En el archivo de proyecto, **Projecto.hip** se añaden las keyword que se deseen, por ejemplo, para la tabla de contenidos y la bibliografía basta con añadir:

- **toc**, para la tabla de contenidos.
- **bibliography**, para la bibliografía.

Y el *ProjectManager* se encargará de añadir la palabra clave como *placeholder* a la lista de documentos, y en los procesamientos que acaban de ser descritos se procesaran también las *keywords*.

6.5.7 Gestión de la bibliografía

Como acaba de mencionarse en el apartado anterior, la bibliografía es una *keyword*, pero su gestión merece una explicación específica. En primer lugar, la bibliografía debe añadirse

como un documento aparte en el directorio *docs*, y debe mantener un formato **bibtex**, que es del tipo:

@Book{MI:CILE:2015, author = {Ministerio de Interior}, title = {Comercio Interior del Libro en España}, year = {2015} }

Para su procesamiento se hace uso de la biblioteca **Bibtex2html**, que, como su nombre indica, se encarga de transformar el documento en formato *bibtex* a *html*. La clase encargada de la gestión de la bibliografía es **ProjectKeywordBibliography**, por un lado invoca a la biblioteca **Bibtex2html**, una vez obtenido el archivo *bibliography*, que es el resultado de la anterior invocación, se procede al procesamiento *html* o *latex*, según proceda. Al igual que sucede con las etiquetas, hay unas plantillas definidas para estos procesamientos.

6.6 Módulo de adaptadores

La utilización de herramientas o tecnologías externas obliga a implementar una serie de adaptadores que permitan su uso dentro del sistema. En este caso han sido varias las herramientas que han debido integrarse:

- Bibtex2Html
- Imagemagick
- Pandoc
- Tex2Im
- Calibre
- Latex
- Unzip

Para cada uno de ellos es necesario crear una clase *adapter*, todos cuentan con tres métodos comunes, y en algunos casos, algunos adicionales.

En primer lugar, el constructor, cuya función es construir una instancia de la interfaz con los ejecutables adecuados en función del Sistema Operativo, poniendo un ejemplo, en el caso de **Unzip**, el constructor quedaría de la siguiente forma:

```
def __init__(self):

    self._comm = os.path.join(strings.Adapters.BIN_DIR, strings.Adapters.UNZIP_DIR,
                              'unzip.exe')

    if sys.platform == 'linux' or sys.platform == 'linux2':
        self._comm = r'unzip'
```

Código 6.19: Constructor adapter

El segundo método común sería run, que ejecuta el comando asignado en el constructor.

```
def run(self, input_file, output_dir):
    command = [self._comm, input_file, '-d', output_dir]
    if self._exec(command) != 0:
        raise HidraRunCommandException('{}'.format(command))
```

Código 6.20: Método run

Y el último método _exec, que ejecuta una instrucción.

Capítulo 7

Conclusiones

En este capítulo se analiza sin se han alcanzado los objetivos planteados en el capítulo 6, además, se proponen una serie de mejoras o tareas pendientes en el subapartado de Propuestas de trabajos futuros.

7.1 Objetivos alcanzados

Tras el desarrollo del sistema que se ha llevado a cabo se han alcanzado con éxito el objetivo general y los objetivos específicos descritos en el capítulo 2. HIDRA es una herramienta de conversión automática enfocada a la edición documental de manuales científico-técnicos.

El objetivo general se ha satisfecho al lograr, proporcionando una estructura de ficheros definida, la producción de dicha estructura, a unos formatos de salida específicos que permitan la generación de libros digitales o su preparación para su posterior publicación en imprenta.

El presente documento es una demostración del alcance de la herramienta HIDRA. En la cual se incluyó el estilo requerido por la Escuela Superior de Informática, y se han integrado todos los elementos desarrollados en el sistema, desde la estructura de capítulos y secciones, pasando por imágenes, tablas, ecuaciones, cuadros de texto… hasta la bibliografía. Todo ello para los distintos formatos que proporciona HIDRA, véase PDF, ePub, azw3, html y SCORM.

Los objetivos específicos también se han alcanzado satisfactoriamente. La definición de un lenguaje propio HIDRA, se ha basado en Markdown, en los anexos hay un breve manual de las funcionalidades de Markdown que se han empleado directamente y de los elementos propios que se han definido, todas las etiquetas que se mencionan en el capítulo de objetivos han sido implementadas. Por otro lado, se ha construido un procesador de lenguaje con análisis léxico y sintáctico, encargado de reconocer estas estructuras. Esto se ha logrado por medio de la herramienta Python-ply, descrito en la subsección Python-ply. El resultado son dos módulos independientes encargados del análisis léxico y sintáctico respectivamente que trabajan de forma cooperativa proporcionando un árbol sintáctico.

La definición de una estructura de proyecto se ha basado en el formato YAML, que se-

rializa en los datos en una colección cómoda de usar y de fácil acceso, en los anexos se ha incluido un archivo de proyecto que muestra un ejemplo de uso del mismo.

Para la automatización de la conversión de elementos de edición se ha hecho uso de distintas herramientas, que de forma colaborativa permiten alcanzar el objetivo. En el caso de las imágenes, se ha hecho uso de ImageMagick, para procesarlas de forma que cada salida cuente con el formato más adecuado. Tex2Im es un módulo de Python que haciendo uso de Pdflatex transforma las ecuaciones en imágenes. Pandoc, por otra parte, se encarga de la automatización de gran número de los elementos de edición restantes, con la conversión del lenguaje Markdown a las distintas salidas.

Continuando con los objetivos específicos, HIDRA se encarga de la gestión de referencias automáticas entre imágenes, tablas, ecuaciones y elementos bibiliográficos en todos los formatos de salida, para ello se ha definido una estructura propia o etiqueta, que para cada tipo de salida, como todas las etiquetas HIDRA, realiza un procesamiento distinto basado en plantillas, que incorporan secciones de código, sustituyendo algunos atributos.

Con el sistema HIDRA se proporcionan distintos estilos, que además son editables, de forma que el usuario pueda personalizarlos a su gusto. Para ello, hay una carpeta en el proyecto denominada *style* donde se incluyen las hojas de estilos: un fichero cls, para la personalización del latex; y otro css, para html y sus derivados (ePub, azw3 y SCORM).

Uno de los objetivos planteados al comienzo del proyecto fue la posibilidad de incluir soporte para código de bajo nivel, por si se desea incluir alguna funcionalidad que no estuviera contemplada en la implementación del sistema. Para lograrlo, se ha hecho uso de una etiqueta de hidra. Es anidada, formada por dos etiquetas, una de principio y otra de fin, en las que se indica el tipo de lenguaje que se va a incluir. Con esto es suficiente para que HIDRA introduzca el segmento de texto añadido en esta etiqueta, para ser procesado sólo en el formato de salida que se ha indicado.

HIDRA puede ser ejecutado tanto para GNU/Linux como para Microsoft Windows, además se desarrolló utilizando estándares y tecnologías libres.

Por último, y para garantizar la facilidad de uso, se proporciona un Manual de uso completo de la herramienta, que incluye secciones como la instalación, para el caso de Microsoft Windows, la creación y gestión de proyectos, y el uso del lenguaje HIDRA.

7.2 Propuestas de trabajo futuros

A pesar de alcanzar los objetivos propuestos, se considera que algunos aspectos aspectos son mejorables, pero por cuestión de tiempo no han podido llevarse a cabo.

A continuación, se describen algunas lineas de trabajo futuro que se preveé mejorarían la herramienta:

- **Mejora del lenguaje HIDRA**:

El lenguaje HIDRA es uno de los pilares principales de la herramienta que se ha desarrollado, y aunque cumple lo requisitos establecidos, se considera que, por ejemplo, podría ser más flexible. Para ello habría que modificar los analizadores léxicos y sintácticos. Si bien es cierto, que desde el comienzo de la implementación del sistema, donde se hacía uso de expresiones regulares para el tratamiento de las etiquetas, hasta ahora, con un procesador del lenguaje, las mejoras se han incrementado de manera exponencial, en muchos casos el lenguaje es un poco estricto, obligando al usuario a adoptar una forma de describir las etiquetas específicas. Por otro lado, no se ha implementado un analizador semántico como tal, si no que el árbol sintáctico obtenido del análisis léxico y sintáctico se proporciona a las estructuras de HIDRA correspondientes, y ellas se encargar de su tratamiento.

El tratamiento de error también es un aspecto a mejorar, ya que hasta el momento, si el procesador detecta un error, el usuario simplemente será informado de que el token es erróneo, en el caso de un error léxico, o un error en la estructura, en el caso de un error sintáctico, indicando el tipo de estructura que se ha añadido.

Se preveé que la mejora de estos aspectos podría llevar unas dos semanas de trabajo, incluyendo tiempo de formación y estudio, y trabajando exclusivamente con las clases *lexico.py* y *yacc.py*.

- **Ampliación de las etiquetas**:

La herramienta esta implementada de tal forma que la inclusión de nuevas etiquetas no debe suponer un gran esfuerzo de implementación. Cada etiqueta consta de una clase, cuyos atributos son las propiedades que definen dicha etiqueta, y unas plantillas para cada formato de salida.

Además, habría que añadir la nueva etiqueta en el diccionario de etiquetas que tiene la clase *Engine*. En el caso excepcional de algunas etiquetas es necesario hacer alguna comprobación durante el procesamiento, pero por regla general, es suficiente con el constructor de la clase de la propia etiqueta.

Se han definido los elementos que se consideran básicos en la publicación multiformato, pero si se desearan añadir nuevas etiquetas, el esfuerzo de implementar una nueva clase con sus respectivas plantillas, y añadir la etiqueta en el diccionario de la clase *Engine*, sería mínimo.

- **Incluir más formatos de salida**:

Como se ha mencionado a lo largo del presente documento, el criterio de elección de los formatos de salida implementados se basa en llegar a un público lo más amplio posible, pero está decisión puede ser subjetiva, o las tendencias pueden cambiar rápidamente, por ello, se considera que la inclusión de nuevos formatos puede ser interesante, aunque también se

preveé más esfuerzo que para las tareas anteriormente mencionadas, pues cada formato de salida realiza un procesamiento específico, además de un estudio del mismo, por lo que se estima que cada nuevo formato podría conllevar unas tres semanas de esfuerzo.

- **Rediseñar la interfaz gráfica**:

Debido a la falta de tiempo se optó por mantener la interfaz inicial, que se caracteriza por ser sencilla y robusta, pero contrapunto es poco elegante. Lo ideal sería una implementación que combinara la sencillez actual con un diseño elegante y fluido. También sería interesante añadir nuevos elementos, como una barra de progreso. Se estima que en una semana podría diseñarse la nueva interfaz e implementarla.

7.3 Conclusión personal

Hace cinco años comenzó la que ha sido, hasta el momento, la etapa más importante de mi vida. Ha sido un camino que no siempre ha resultado agradable ni fácil, ha supuesto trabajar duro y largas noches sin dormir, pero sin duda ha merecido la pena. Con los años, mis compañeros y yo, hemos ido adquiriendo conocimientos sobre muy diversas áreas, y hemos aprendido a aplicarlos, pero es en proyectos como este en los que hay que demostrar que todo lo aprendido ha dejado huella, que se es capaz de encontrar las herramientas necesarias para afrontar los problemas que vayan surgiendo. A pesar de todas las prácticas realizadas durante la carrera, en ningún caso he abordado algo tan complejo como el sistema desarrollado en mi trabajo fin de grado. Y estoy orgullosa de lo que he conseguido. Me he esforzado y he alcanzado lo que me proponía, y si tuviera que volver a hacerlo, escogería de nuevo esta carrera que tanto me ha dado, y quitado, y que me definirá desde ahora y para siempre.

ANEXOS

Anexo A: Requisitos

Objetivo: Proporcionar una herramienta de conversión automática que, partiendo de un documento especificado en un formato tipo Markdown, obtenga una salida en diversos formatos para publicación. En principio, se plantea el soporte para los siguientes tipos de archivo de salida: PDF (maquetado para impresión o publicación online, generado vía LaTeX), Epub (Ebook multidispositivo), Mobi (Amazon Kindle), Scorm (integración con Moodle).

Resumen de Especificación Técnica: El sistema contará con un archivo de especificación de proyecto y un árbol de directorios donde se especificarán los recursos gráficos y de texto a utilizar. El sistema tomará como entrada ese archivo y generará los documentos de salida.

```
#Configuracion general
images = imgs/       # Directorio de donde cuelgan las imágenes (con subdir)
documents = docs/    # Directorio donde están los documentos Markdown de entrada
workdir = temp/      # Directorio donde se generan los temporales (imágenes...)
output = out/        # Directorio donde se generan las salidas
pdfsize = a5         # Tamaño específico de salida en PDF
toclevel = 2         # Profundidad de la tabla de contenidos
style = book         # Estilo elegido para generar el documento

#Título e información del proyecto
title = "Título del libro"
author = "Perico Pérez"
version = 1.0
cover = cover.jpg   # Relativo a carpeta de images
abstract = abstract.md  # Relativo a directorio de documents

#Lista de documentos (si no tienen extensión, es autogenerado)
abstract.md
prefacio.md
toc                 # Palabra clave para generar la tabla de contenidos
documento1.md
documento2.md
bibliography        # Palabra clave para generar la bibliografía
```

Figura 1: Ejemplo de archivo de proyecto (Proyecto.hip)

Soporte de tags requerido

- **Capítulo**. En salida PDF será un *chapter* de LaTeX. En Epub y similar será un capítulo independiente.
- **Encabezado de nivel 1-4**. Generarán entradas en tabla de contenidos. Se numerarán hasta el nivel indicado en el archivo de configuración.
- **Estilo de texto**. Normal, **negrita**, *cursiva*.

- **Imágenes**. Insertadas a ancho completo o en línea (ver "Tags de Hidra para formato de salida").
- **Terminal**. Estilo monoespaciado para comandos y órdenes incluidas en el flujo del texto.
- **Tablas**. Usaremos los formatos por defecto de Markdown.

Tags de Hidra para un formato de salida

- **Imágenes**. El tag de imagen soportará varios tipos de imagen, opcionalmente una etiqueta, una referencia y la inclusión en varios formatos. El formato será algo parecido a:

  [img][file][100]["Descripción de la imagen"][imgMapa] # Imagen ancho total

  [imgin][file] # Imagen de tipo inline

- La imagen insertada en el texto lo hará en un tamaño proporcional al ancho de documento (tercer parámetro indica porcentaje sobre ancho de bloque).
- En "file" no se indicará la extensión del archivo. Si existe la imagen en formato .pdf, se utilizará en la salida LaTeX (para no perder información vectorial si la hubiera). A partir de este formato se generará automáticamente la versión .jpg para formatos de salida que no soporten la inclusión en PDF. Si no existe, se buscará la imagen en .jpg.
- El último tag servirá para la inclusión de referencias cruzadas. Esta generación tendrá que ser realizada en preproceso.
- **Tablas**. Se utilizará el formato de tablas de Markdown, pero deberá soportar igualmente etiquetas y referencias. Tras la defínción de una tabla, se podrá incluir un tag del tipo:

  [tab]["Descripción de la tabla"][tabPrecios]

- **Bibliografía**. Idealmente habría que soportar bibliografía en algún formato estándar, como BibTeX. La idea es que se generen referencias válidas que puedan ser utilizadas en el comando de "Referencias". Existirá un comando específico para indicar dónde se genera la sección de bibliografía.
- **Referencias**. Existirá un tag específico para referenciar imágenes y tablas. El uso sería algo similar a:
- **Links**. Existirá un tag propio para enlaces. El último parámetro (opcional) indica si la URL se imprimirá también en el documento o aparecerá oculto (implícito en el link).

  [link][http://www.google.es]["Web de google"][showurl]

- **Código**. Soporte para especificar fragmentos de código. Posibilidad (opcional) de incluir lenguaje del fragmento.
- **Ecuaciones**. Soporte para ecuaciones en formato LaTeX. Pueden ser inline o completas. En formatos basados en HTML se renderizarán a imagen.

Soporte para tags de bajo nivel

Posibilidad de incrustar fragmentos específicos para cada plataforma de publicación. Se definirán tags de inicio y fin de cada formato.

Anexo B: Crear un nuevo proyecto

El primer paso para maquetar los documentos deseados es crear un nuevo proyecto.

Al abrir HIDRA se muestra la siguiente pantalla:

Figura 2: Pantalla de inicio

Pulsar en *Nuevo*, se abrirá la pantalla de nuevo proyecto:

Figura 3: Nuevo proyecto

Escribir un nombre para el proyecto y seleccionar la ruta donde crearlo. Una vez hecho esto pulsar en *Generar proyecto*. Si todo se ha realizado correctamente se abrirá la carpeta con el proyecto:

Automáticamente se genera una estructura de carpetas que cuenta con los siguientes elementos:

- **docs**: En esta carpeta se añadirán los documentos con extensión *.md* que compondrán el proyecto.

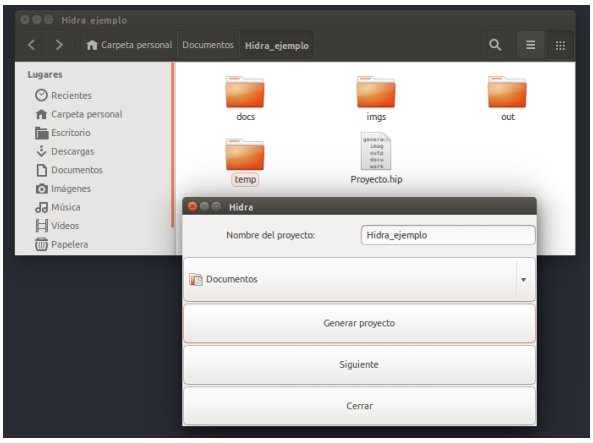


Figura 4: Carpeta de proyecto

- **imgs**: Contendrá todas las imágenes a las que se hace referencia en los documentos. Por defecto contiene las imágenes de los cuadros de texto.
- **out**: Aquí se crearán distintas carpetas para cada uno de los formatos de salida seleccionados, con sus versiones finales.
- **temp**: Carpeta donde se irán añadiendo los temporales necesitados por HIDRA.
- **Proyecto.hip**: Es el documento principal del proyecto, con extensión *.hip* (HIDRA PROYECT), proporciona a HIDRA información sobre distintos aspectos del proyecto, a continuación se muestra un ejemplo:

```
general:
  images: imgs/
  output: out/
  documents: docs/
  workdir: temp/
  toclevel: 2
  style: amazon
  pdfsize: a4

project:
  abstract: abstract.md
  version: 1.0
  author: "Nombre del autor"
  title: "Titulo del libro"
  cover: cover.jpg

documents:
  list:
  - cover
  - abstract.md
  - prefacio.md
  - toc
  - documento1.md
  - documento2.md
  - bibliography
```

Figura 5: Archivo Proyecto.hip

En **general** se seleccionar las carpetas donde buscar o crear los distintos elementos del proyecto, además de poder seleccionar en *toc* el nivel de anidación de la tabla de contenidos, en *style* el estilo a utilizar, y *pdfsize* el tamaño del PDF.

En **project** se indica que archivo contiene el *abstract* si existe, el *autor*, el *titulo* del proyecto y la *portada*.

Por último, **documents** contiene una lista de los documentos que componen el proyecto, ordenados según se desea que aparezcan en los formatos finales.

El usuario debe modificar el documento **Proyecto.hip** para que se ajuste a sus necesidades, y añadir en **documents** e **imgs** los documentos e imágenes necesarios, respectivamente. Una vez hecho esto se procede a realizar el último paso:

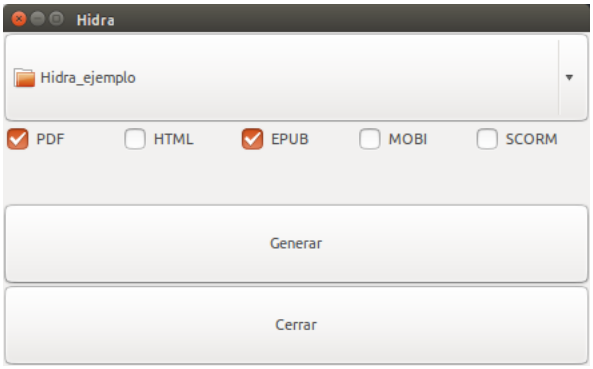


Figura 6: Generación de los formatos finales

Marcar los formatos de salida deseados, y pulsar **Generar**.

Si todo se ha realizado correctamente, cuando HIDRA haya terminado de procesar se abrirá la carpeta que contiene los formatos de salida seleccionados.

Anexo C: Abrir un proyecto

No es necesario volver a crear un proyecto cada vez que se procesen los documentos de un proyecto, para ello existe la opción **Abrir** cuando se abre la pantalla inicial de HIDRA:

Figura 7: Pantalla de inicio

El botón **Abrir** mostrará la pantalla de generación de documentos:

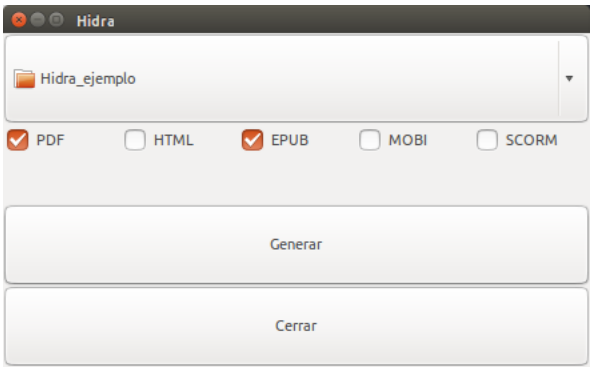


Figura 8: Generación de los formatos finales

Abrir la pestaña para seleccionar la ruta del proyecto, marcar con un tick para obtener los formatos de salida deseados, y pulsar **Generar**.

Si todo se ha realizado correctamente, cuando HIDRA haya terminado de procesar se abrirá la carpeta que contiene los formatos de salida seleccionados.

Anexo D: Sintaxis de etiquetas propias

Para dar soporte a las distintas necesidades que iban teniendo lugar a la hora de maquetar documentos se han definido distintos tipos de etiquetas, a continuación se muestra una pequeña descripción y ejemplo de uso de cada una de ellas.

Imágenes

El sistema HIDRA cuenta con tres tipos de imágenes hasta el momento.

Imágenes simples

Las imágenes de este tipo se situarán centradas en la página. Como parámetros aceptan la *ruta* de la imágen, un tanto por ciento que representa el *ancho* de la imágen en proporción a la página, *descripción* de la imágen, *referencias* a otras etiquetas, *identificador* de la etiqueta para que sea referenciable.

Figura 9: Ejemplo de imagen

La descripción de la imágen y las referencias a otras etiquetas son opcionales.

La sintaxis que emplea es la siguiente:

[img, path=ogre.jpg, width=25, label="Ejemplo de imagen", ref=EjemploImg]

Imágenes in line

Las imágenes in line se sitúan en línea  con el texto en el que se añaden, adaptando su tamaño al tamaño de la fuente empleada.

Sólo acepta la *ruta* de la imágen.

La sintaxis que emplea es la siguiente:

[imgin, path=ogre.jpg]

Imágenes laterales

Las imágenes laterales son una implementación exclusiva para la maquetación en pdf. Estas imágenes se situarán en el lateral de la página, y la descripción se situará al lado.

Figura 10: Ejemplo de imagen lateral

Acepta los mimos atributos que las imágenes simples.

La sintaxis que emplea es la siguiente:

[imglat, path=ogre.jpg, width=25, label="Ejemplo de imagen lateral", ref=EjemploImgLat]

Tablas

Las tablas siguen el formato Markdown, y además se le añaden las características definidas en la etiqueta hidra.

Markdown	Less	Pretty
Still	`renders`	**nicely**
1	2	3

Cuadro 1: Esta tabla es el primer ejemplo 11.1

Pueden indicarse una descripción de la tabla, un identificador con el que hacerle referencia, y como en el resto de las etiquetas, referencias a otras etiquetas.

La sintaxis que usa es la siguiente:

[tab, ref=tabla1, label="Esta tabla es el primer ejemplo", [ref, ref=EjemploImg]]

Debe añadirse después de la tabla en formato *Markdown*.

Enlaces

Los atributos que acepta son: la **URL**, el **título** o **nombre** que aparecerá como link, y si se **muestra** o no la dirección web.

[link, url=http://blog.uclm.es/cted/proyectos/hidra/, title=HIDRA, show=no]

Los enlaces a URL's siguen la siguiente sintaxis:

[link, url=http://blog.uclm.es/cted/proyectos/hidra/, title=HIDRA, show=no]

Referencias

Las referencias pueden añadirse en cualquier lugar del texto, pero también como atributo dentro de las etiquetas HIDRA, por ejemplo:

Tabla Esta tabla es el primer ejemplo 11.1

La sintaxis que emplea es:

> [ref, ref=tabla1]

Donde *ref* indica a qué se hace referencia.

Código

Para insertar secciones de código se utilizan las etiquetas *codebegin* y *codeend*, que se añadirán al comienzo y al final del código.

```
System.out.println("Print de prueba")
int hola=8;
float adios = 8.4f;
```

Anexo D: Prueba con codigo de ejemplo

Los atributos que acepta la etiqueta *codebegin* son el **lenguaje de programación**, el **identificador** con el que hacer referencia, **descripción** del listado de código, y **referencia** a otras etiquetas.

La sintaxis que emplearía para la etiqueta de comienzo sería:

> [codebegin, language=java, ref=codigoJava, label="Prueba con codigo de ejemplo"]

Y para la etiqueta de final:

> [codeend]

Entre ambas etiquetas se añade el código.

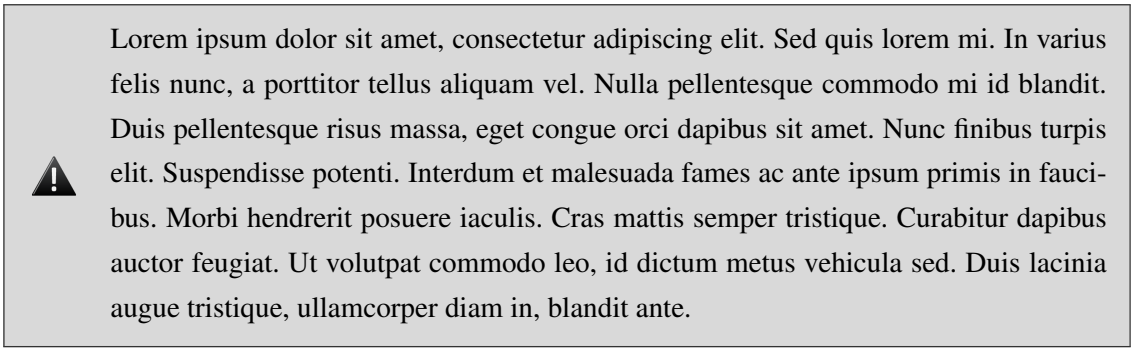
Cuadros de texto

Existen tres opciones disponibles para los cuadros de texto:

- Información
- Pregunta

- Alerta

La imagen que acompaña al texto dentro de cada cuadro cambiará de acuerdo a la opción seleccionada.



La sintaxis que emplea consiste en dos etiquetas, una al comienzo del texto y otra al final.

[textboxbegin, icon=warning]

El atributo icon puede tomar como valor *warning*, *info* y *question*.

[textboxend]

Es la etiqueta que se añade al final del texto que contendrá el cuadro.

Ecuaciones

Las ecuaciones siguen la misma sintaxis que *Latex*, pero antes y después de las ecuaciones hay que añadir una etiqueta de comienzo y de final respectivamente:

$$\frac{p'_x}{p_x} = \frac{-d}{p_z} \quad \Leftrightarrow \quad p'_x = \frac{-dp_x}{p_z} \tag{1}$$

La sintaxis que emplearía sería:

[eqbegin, ref=eq]

Como etiqueta de comienzo, donde *ref* es el nombre con el que se hará referencia a la ecuación.

[eqend]

Como etiqueta de final.

Código de bajo nivel

Las etiquetas de código de bajo nivel constituyen un caso especial, para solventar necesidades que aún no hayan sido implementadas.

Añadiento estas etiquetas se insertan secciones de código en un lenguaje determinado, que solo se interpretarán por el formato seleccionado. Es decir, si se desea añadir código html, con esta etiqueta el procesador del resto de formatos de salida lo pasará por alto, y solo se mostrará en html el código que se ha añadido.

Anexo E: Sintaxis de Markdown

Como base en la conversión de los documentos del proyecto se hace uso de *Markdown*, que es un lenguaje de marcado que permite aplicar cierto formato a un texto, de forma muy sencilla, a continuación se muestran ejemplos de los elementos más comunes de los que se hará uso.

Cabeceras

Markdown ofrece una jerarquía de cabeceras, con los que se representarán los capítulos, secciones y subsecciones.

Para ello, se debe añadir '#' antes del titulo de la cabecera. El número de '#' que se añada representa el nivel de la cabecera. En la imágen 12.1 pueden verse ejemplos de cabeceras y su conversión.

Figura 11: Cabeceras en Markdown y su representación

La cabeceras de nivel 1 representan los capítulos, las de nivel 2 las secciones, y según aumenta el número de los niveles se introduce un nivel dentro de las subsecciones.

Párrafos

Si se desea crear un párrafo basta con dejar una línea en blanco.

Y para crear un salto de línea dentro de un párrado se dejan dos espacios
alfinal de la última palabra de la línea.

Formato negrita y cursiva

Hay distintas formas de representar el formato negrita, cursiva o ambas.

- *Cursiva:* encerrar el texto entre asteriscos simples *Ejemplo*
- *Cursiva:* encerrar el texto entre guiones bajos simples _Ejemplo_
- **Negrita:** encerrar el texto entre parejas de asteriscos **Ejemplo**
- **Negrita:** encerrar el texto entre parejas de guines bajos __Ejemplo__
- ***Negrita y cursiva:*** encerrar el texto entre asteriscos triples ***Ejemplo***
- ***Negrita y cursiva:*** encerrar el texto entre guines bajos triples ___Ejemplo___

Citas

Para citas basta con escribir el símbolo mayor que > antes del texto.

> Es tu diseño es tu decisión.

Pueden crearse bloques anidados de citas añadiendo más >

> Es tu diseño
>
> > Es tu decisión.

Listas

Para crear listas ordenadas hay que añadir el número correspondiente seguido de un punto antes del item de la lista.

1. Elemento 1 de lista ordenada
2. Elemento 2 de lista ordenada
3. Elemento 3 de lista ordenada

Si las listas son desordenadas, en lugar de un número se añadirán los símbolos *, +, - indistintamente.

- Elemento 1 de lista desordenada
- Elemento 2 de lista desordenada
- Elemento 3 de lista desordenada

Notas a pie de página

Las notas a pie de página[1] se componen de dos elementos, el superíndice en la palabra que se quiere referenciar en el pie de página, y la definición que se mostrará al final de la página, esta última debe colocarse al final de la última página de los documentos del proyecto.

La sintaxis que usa es:

[1] El texto de la nota al pie de página

- *[^1]* para el superíndice, donde el número 1 será sustituido por el correspondiente en cada caso.
- *[^1]: El texto de la nota al pie de página*; para la definición de nota al pie de página.

Anexo F: Analizador léxico

```
import ply.lex as lex
from exceptions import HidraTagParserErrorException

tokens = (
                'LCORCHETE',
                'RCORCHETE',
                'COMA',
                'IGUAL',
                'IDENTIFICADOR',
                'SIMBOLOS',
                'LITERAL'
        )

t_LCORCHETE     = r'\['
t_RCORCHETE     = r']'
t_COMA          = r'\,'
t_IGUAL         = r'='
t_IDENTIFICADOR = r'[a-zA-Z:]+'
t_SIMBOLOS      = r''
t_LITERAL       = r''

t_ignore  = ' \t'

def t_error(token):
    message = "Token desconocido:"
    message = "\ntype:" + token.type
    message += "\nvalue:" + str(token.value)
    message += "\nline:" + str(token.lineno)
    message += "\nposition:" + str(token.lexpos)
    raise HidraTagParserErrorException(message)

lex.lex()

if __name__ == '__main__':
    lex.runmain()
```

Anexo F: Analizador léxico

Anexo G: Analizador sintáctico

```
import ply.yacc as yacc
from lexico import tokens

DEBUG = True

def add(e,l):
    l.insert(0,e)
    return l

def p_tag(p):
    'tag : LCORCHETE IDENTIFICADOR lista RCORCHETE'
    p[0] = [p[2],p[3]]

def p_atributo_elemento(p):
    'atributo : elemento lista'
    p[0] = add(p[1], p[2])

def p_atributo_tag(p):
    'atributo : tag lista'
    p[0] = add([p[1]], p[2])

def p_lista(p):
    'lista : COMA atributo'
    p[0] = p[2]

def p_lista_empty(p):
    'lista : empty'
    p[0] = []

def p_empty(p):
    'empty :'
    pass

def p_elemento(p):
    'elemento : IDENTIFICADOR IGUAL opciones'
```

```
    p[0] = [p[1], p[3]]

def p_opciones_cadena(p):
    'opciones : cadena'
    p[0] = p[1]

def p_opciones_literal(p):
    'opciones : LITERAL'
    p[0] = p[1]

def p_cadena_t(p):
    'cadena : IDENTIFICADOR cadena'
    p[0] = p[1]+p[2]

def p_cadena_s(p):
    'cadena : SIMBOLOS cadena'
    p[0] = p[1]+p[2]

def p_cadena_empty(p):
    'cadena : empty'
    p[0] = ""

def p_error(p):
    print "Syntax error ",p

yacc.yacc()
```

Anexo G: Analizador sintáctico

Anexo H: Código Fuente

El código fuente completo del sistema no se ha añadido debido a la extensión del mismo, sin embargo, se ha incluído en el DVD que se adjunta al presente documento.

A continuación se describe la estructura de directorios de la misma:

- **bin/**: Contiene los binarios de las herramientas necesarias para la ejecución del sistema. A continuación se muestra la lista de directorios que contiene, cada uno de los cuales contiene los binarios de la herramienta que los nombra.
  - **bibtex2html/**
  - **calibre/**
  - **imagemagick/**
  - **latex/**
  - **pandoc/**
  - **unzip/**
- **data/**: Incluye dos subdirectorios con información relativa a las plantillas de estilos y ejemplos de uso.
  - **samples/**: Esta carpeta contiene un ejemplo de uso que ilustra todas las etiquetas definidas.
  - **templates/**: Contiene las plantillas de estilo de todas las etiquetas asociadas a los distintos estilos.
    - **amazon/**
    - **article/**
    - **book/**
    - **images/**
    - **scorm/**
    - **tfg/**
- **etc/**: Archivos necesario para la ejecución de gtk-2.0
- **glade/**: Contiene algunos archivos necesarios para la interfaz gráfica
- **hidra/**: Este directorio contiene el núcleo de la aplicación

- **adapters/**: Contiene los adaptores que invocan a las distintas herramientas, según el sistema.
- **keywords/**: Directorio que incluye los archivos encargados del procesamiento de las *keywords*, palabras que hacen referencia a elementos como *bibliografía, portada,* o *tabla de contenido* entre otros.
- **tags/**: Este directorio contiene las clases de las etiquetas implementadas.
- **tasks/**: Incluye clases *.py* encargadas de las tareas principales del sistema, como el *preprocesamiento* o *procesamiento latex*.
- **test/**: Contiene test para probar funcionalidades del sistema.

- **lib/**: Archivos necesario para la ejecución de gtk-2.0
- **share/**: Archivos autogenerados necesarios para la ejecución de gtk-2.0
- **wget/**: Binarios para la ejecución en windows

Anexo I: Manual de Instalación Windows

Para el uso de HIDRA en Windows se ha generado un instalador, a continuación se muestran unas capturas del proceso de instalación.

El primer paso es la selección del idioma, véase figura 16.1.

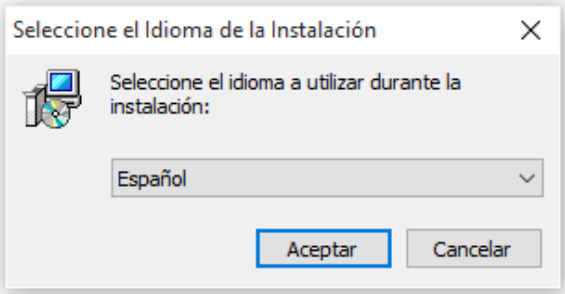


Figura 12: Ventana de selección de idioma

A continuación, la siguiente ventana nos pedirá una confirmación del acuerdo de licencia (figura 16.2).

Seleccionar, si se desea, alguna de las tareas adicionales, en este caso, *Crear un icono en el escritorio*.

En la imágen 16.4 se muestra una captura del icono en el escritorio.

En el siguiente paso simplemente confirmar la instalación, véase 16.5.

En la figura 16.6 puede verse el progreso de la instalación.

Por último, la última ventana de la instalación, figura 16.7 permite lanzar la aplicación. Y el proceso de instalación habría terminado.

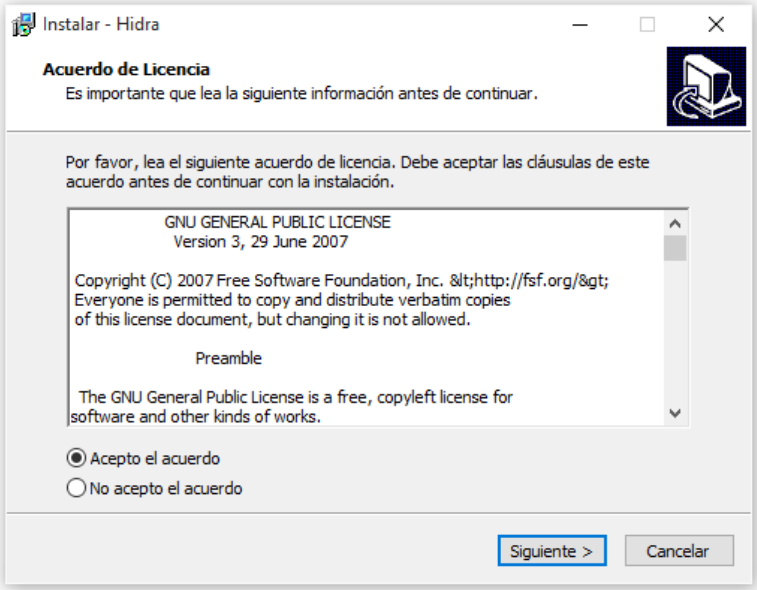


Figura 13: Confirmación del acuerdo de licencia

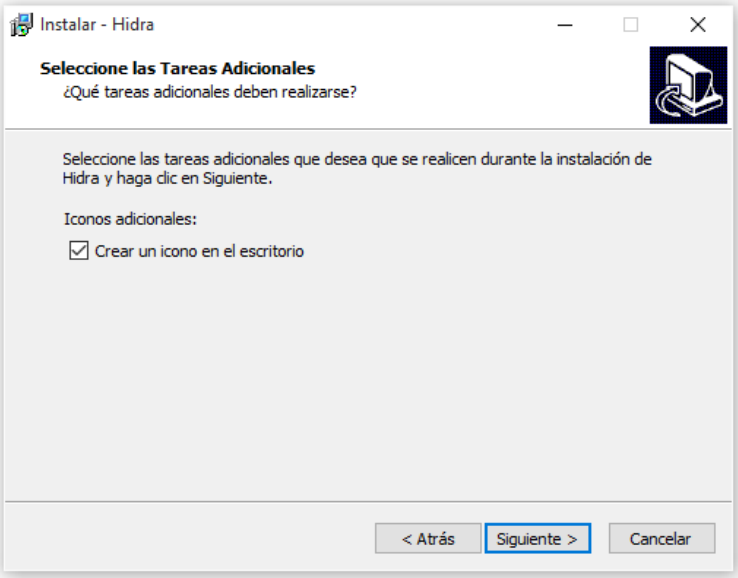


Figura 14: Selección de tareas adicionales

Figura 15: Acceso directo en el escritorio

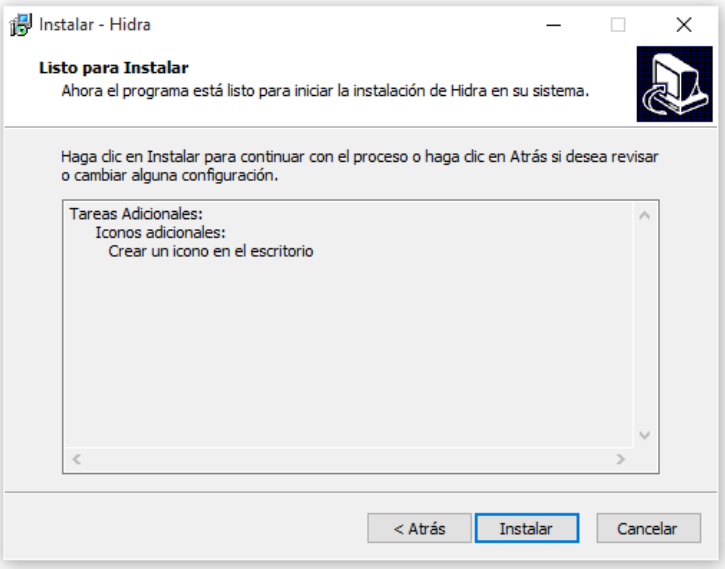


Figura 16: Confirmación de instalación

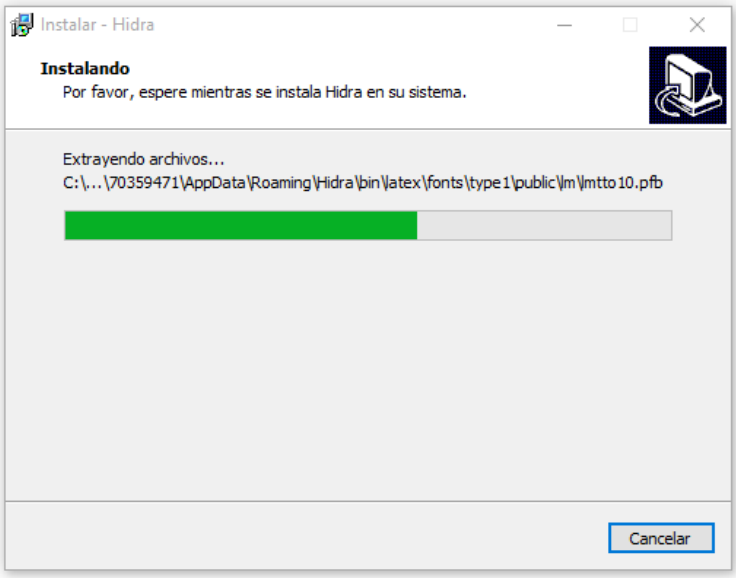


Figura 17: Progreso de la instalación

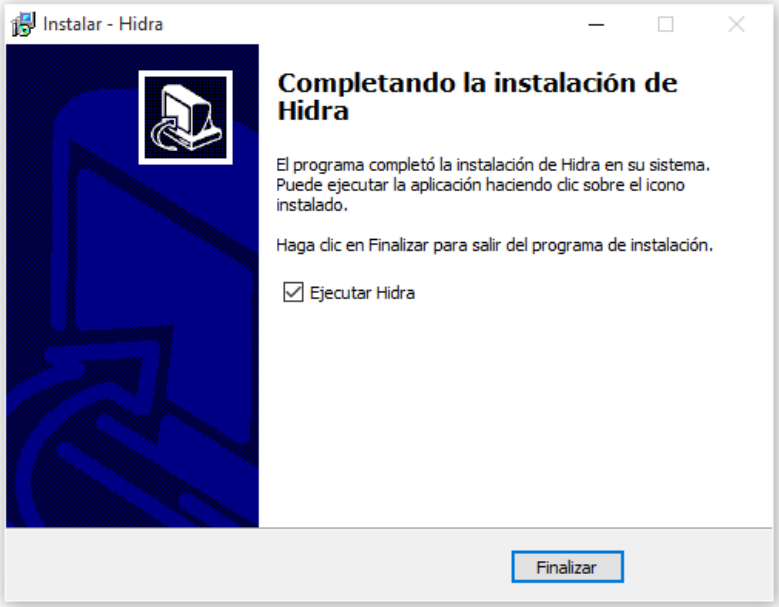


Figura 18: Finalización del proceso de instalación

Anexo J: Salida generada por el parser

```
Created by PLY version 3.8 (http://www.dabeaz.com/ply)

Grammar

Rule 0     S' -> tag
Rule 1     tag -> LCORCHETE IDENTIFICADOR lista RCORCHETE
Rule 2     atributo -> elemento lista
Rule 3     atributo -> tag lista
Rule 4     lista -> COMA atributo
Rule 5     lista -> empty
Rule 6     empty -> <empty>
Rule 7     elemento -> IDENTIFICADOR IGUAL opciones
Rule 8     opciones -> cadena
Rule 9     opciones -> LITERAL
Rule 10    cadena -> IDENTIFICADOR cadena
Rule 11    cadena -> SIMBOLOS cadena
Rule 12    cadena -> empty

Terminals, with rules where they appear

COMA                 : 4
IDENTIFICADOR        : 1 7 10
IGUAL                : 7
LCORCHETE            : 1
LITERAL              : 9
RCORCHETE            : 1
SIMBOLOS             : 11
error                :

Nonterminals, with rules where they appear

atributo             : 4
cadena               : 8 10 11
elemento             : 2
empty                : 5 12
```

```
lista                : 1 2 3
opciones             : 7
tag                  : 3 0

Parsing method: LALR

state 0

    (0) S’ -> . tag
    (1) tag -> . LCORCHETE IDENTIFICADOR lista RCORCHETE

    LCORCHETE       shift and go to state 1

    tag                           shift and go to state 2

state 1

    (1) tag -> LCORCHETE . IDENTIFICADOR lista RCORCHETE

    IDENTIFICADOR   shift and go to state 3


state 2

    (0) S’ -> tag .



state 3

    (1) tag -> LCORCHETE IDENTIFICADOR . lista RCORCHETE
    (4) lista -> . COMA atributo
    (5) lista -> . empty
    (6) empty -> .

    COMA            shift and go to state 4
    RCORCHETE       reduce using rule 6 (empty -> .)

    lista                         shift and go to state 5
    empty                         shift and go to state 6

state 4

    (4) lista -> COMA . atributo
    (2) atributo -> . elemento lista
    (3) atributo -> . tag lista
    (7) elemento -> . IDENTIFICADOR IGUAL opciones
```

```
    (1) tag -> . LCORCHETE IDENTIFICADOR lista RCORCHETE

    IDENTIFICADOR   shift and go to state 8
    LCORCHETE       shift and go to state 1

    tag                           shift and go to state 9
    atributo                      shift and go to state 7
    elemento                      shift and go to state 10

state 5

    (1) tag -> LCORCHETE IDENTIFICADOR lista . RCORCHETE

    RCORCHETE       shift and go to state 11


state 6

    (5) lista -> empty .

    RCORCHETE       reduce using rule 5 (lista -> empty .)


state 7

    (4) lista -> COMA atributo .

    RCORCHETE       reduce using rule 4 (lista -> COMA atributo .)


state 8

    (7) elemento -> IDENTIFICADOR . IGUAL opciones

    IGUAL           shift and go to state 12


state 9

    (3) atributo -> tag . lista
    (4) lista -> . COMA atributo
    (5) lista -> . empty
    (6) empty -> .

    COMA            shift and go to state 4
    RCORCHETE       reduce using rule 6 (empty -> .)
```

```
    lista                          shift and go to state 13
    empty                          shift and go to state 6

state 10

    (2) atributo -> elemento . lista
    (4) lista -> . COMA atributo
    (5) lista -> . empty
    (6) empty -> .

    COMA            shift and go to state 4
    RCORCHETE       reduce using rule 6 (empty -> .)

    lista                          shift and go to state 14
    empty                          shift and go to state 6

state 11

    (1) tag -> LCORCHETE IDENTIFICADOR lista RCORCHETE .

    COMA            reduce using rule 1 (tag -> LCORCHETE IDENTIFICADOR lista RCORCHETE .)
    RCORCHETE       reduce using rule 1 (tag -> LCORCHETE IDENTIFICADOR lista RCORCHETE .)
    $end            reduce using rule 1 (tag -> LCORCHETE IDENTIFICADOR lista RCORCHETE .)


state 12

    (7) elemento -> IDENTIFICADOR IGUAL . opciones
    (8) opciones -> . cadena
    (9) opciones -> . LITERAL
    (10) cadena -> . IDENTIFICADOR cadena
    (11) cadena -> . SIMBOLOS cadena
    (12) cadena -> . empty
    (6) empty -> .

    LITERAL         shift and go to state 18
    IDENTIFICADOR   shift and go to state 16
    SIMBOLOS        shift and go to state 19
    COMA            reduce using rule 6 (empty -> .)
    RCORCHETE       reduce using rule 6 (empty -> .)

    cadena                         shift and go to state 17
    empty                          shift and go to state 20
    opciones                       shift and go to state 15

state 13
```

```
    (3) atributo -> tag lista .

    RCORCHETE       reduce using rule 3 (atributo -> tag lista .)


state 14

    (2) atributo -> elemento lista .

    RCORCHETE       reduce using rule 2 (atributo -> elemento lista .)


state 15

    (7) elemento -> IDENTIFICADOR IGUAL opciones .

    COMA            reduce using rule 7 (elemento -> IDENTIFICADOR IGUAL opciones .)
    RCORCHETE       reduce using rule 7 (elemento -> IDENTIFICADOR IGUAL opciones .)


state 16

    (10) cadena -> IDENTIFICADOR . cadena
    (10) cadena -> . IDENTIFICADOR cadena
    (11) cadena -> . SIMBOLOS cadena
    (12) cadena -> . empty
    (6) empty -> .

    IDENTIFICADOR   shift and go to state 16
    SIMBOLOS        shift and go to state 19
    COMA            reduce using rule 6 (empty -> .)
    RCORCHETE       reduce using rule 6 (empty -> .)

    cadena                        shift and go to state 21
    empty                         shift and go to state 20

state 17

    (8) opciones -> cadena .

    COMA            reduce using rule 8 (opciones -> cadena .)
    RCORCHETE       reduce using rule 8 (opciones -> cadena .)


state 18

    (9) opciones -> LITERAL .
```

```
    COMA            reduce using rule 9 (opciones -> LITERAL .)
    RCORCHETE       reduce using rule 9 (opciones -> LITERAL .)


state 19

    (11) cadena -> SIMBOLOS . cadena
    (10) cadena -> . IDENTIFICADOR cadena
    (11) cadena -> . SIMBOLOS cadena
    (12) cadena -> . empty
    (6) empty -> .

    IDENTIFICADOR   shift and go to state 16
    SIMBOLOS        shift and go to state 19
    COMA            reduce using rule 6 (empty -> .)
    RCORCHETE       reduce using rule 6 (empty -> .)

    cadena                      shift and go to state 22
    empty                       shift and go to state 20

state 20

    (12) cadena -> empty .

    COMA            reduce using rule 12 (cadena -> empty .)
    RCORCHETE       reduce using rule 12 (cadena -> empty .)


state 21

    (10) cadena -> IDENTIFICADOR cadena .

    COMA            reduce using rule 10 (cadena -> IDENTIFICADOR cadena .)
    RCORCHETE       reduce using rule 10 (cadena -> IDENTIFICADOR cadena .)


state 22

    (11) cadena -> SIMBOLOS cadena .

    COMA            reduce using rule 11 (cadena -> SIMBOLOS cadena .)
    RCORCHETE       reduce using rule 11 (cadena -> SIMBOLOS cadena .)
```

Anexo J: Salida generada por el parser

Anexo K: Clase abstracta HidraTask

```
import abc

import logging
log = logging.getLogger('hidra')


class HidraTask(object):
    __metaclass__ = abc.ABCMeta

    def __init__(self, projectManager):
        self._projectManager = projectManager

    @property
    def projectManager(self):
        return self._projectManager

    @abc.abstractmethod
    def run(self):
        pass
```

Anexo K: Clase abstracta HidraTask

Anexo L: Código fuente de ProjectManager

```
from . import utils, strings, keywords
from .project import Project
from .exceptions import HidraProjectNotLoadedException

import os
import shutil

import logging
log = logging.getLogger('hidra')


class ProjectManager(object):
    _PROJECT_KEYWORDS = {
        strings.Project.TOC: keywords.ProjectKeywordTOC,
        strings.Project.BIBLIOGRAPHY: keywords.ProjectKeywordBibliography,
        strings.Project.COVER: keywords.ProjectKeywordCover,
        strings.Project.FRONT: keywords.ProjectKeywordFront,
        strings.Project.MAIN: keywords.ProjectKeywordMain
    }

    def __init__(self):
        self._project = None
        self._documents = []
        self._keywords = {}

    @property
    def project(self):
        return self._project

    @property
    def documents(self):
        return self._documents

    @property
    def keywords(self):
```

```
    return self._keywords

def generateBibliographyHtml(self):
    keywords.ProjectKeywordBibliography(self).processBibliography()

def createProject(self, directory, name, openInFileManager):
    """Crea un nuevo proyecto con la estructura de archivos original"""

    self._project = Project(os.path.join(directory, name))

    try:
        utils.makeDirSafe(self.project.projectDir)
    except OSError as e:
        if e.errno == errno.EEXIST:
            raise HidraProjectAlreadyExistsException(e)
        raise

    shutil.copy(os.path.join(self.project.templatesDir, 'default_project.hip'),
                os.path.join(self.project.projectDir, strings.Project.PROJECT_FILENAME))

    utils.restartLogger(log, os.path.join(self.project.projectDir,
                                          strings.Internal.LOG_FILE))
    log.info('[PROYECTO] Creando un proyecto nuevo en "%s"', self.project.projectDir)

    self.openProject(self.project.projectDir)
    self._makeDirectories()

    if openInFileManager:
        utils.openDir(self.project.projectDir)

    return self.project.projectDir

def openProject(self, directory):
    """Abre un proyecto existente. Ademas, lee todos los documentos y
    palabras clave, y las almacena."""

    if not self._project:
        self._project = Project(directory)

    utils.restartLogger(log, os.path.join(self.project.projectDir,
                                          strings.Internal.LOG_FILE))
    log.info('[PROYECTO] Abriendo el proyecto "%s"', directory)

    self._keywords = {}
    self._documents = []
    hip = self.project.load()
    entries = hip[strings.Project.DOCUMENTS_SECTION][strings.Project.LIST]
```

```
        for index, d in enumerate(entries):
            if d in self._PROJECT_KEYWORDS:
                # Es una palabra clave: crearla para su posterior procesado
                self._keywords[d] = self._PROJECT_KEYWORDS[d](self)
                self._documents.append(d)
            else:
                # Es un documento: almacenar su ruta para concatenarlo despues
                path = os.path.join(self.project.projectDir,
                                    self.project.documentsDir(),
                                    d)
                self._documents.append(path)

        return self.project.projectDir

    def reloadStyle(self):
        """Tarea que copia las plantillas (estilos e imagenes) al directorios
        del proyecto"""
        list = os.listdir(os.path.join(self.project.templatesDir, self.project.style()))
        styles = []
        style_latex = ""
        cls_latex = []
        for l in list:
            stylename = l.split('.')
            if stylename[1] == 'his':
                styles.append(os.path.join(self.project.templatesDir, self.project.style(), l))
            elif stylename[1] == 'cls':
                style_latex = os.path.join(self.project.templatesDir, self.project.style(), l)
            elif stylename[1] == 'sty':
                cls_latex.append(os.path.join(self.project.templatesDir, self.project.style(), l))
        utils.concatDocuments(styles, os.path.join(self.project.inputDirStyles(),
                                                   'style.css'))
        shutil.copy(style_latex, os.path.join(self.project.inputDirStyles()))

        images = os.listdir(self.project.templatesImagesDir)
        for img in images:
            shutil.copy(os.path.join(self.project.templatesImagesDir, img),
                        self.project.imagesDir())

        for c in cls_latex:
            shutil.copy(c, self.project.inputDirStyles())

    def reloadProject(self):
        """Recarga el proyecto. Llamar cuando se modifique el arbol
        de archivos"""

        return self.openProject(self.project.projectDir)
```

```
def _makeDirectories(self):
    """Crea todos los directorios necesarios para el proyecto"""

    dirs = [self.project.imagesDir(),
            self.project.documentsDir(),
            self.project.workDir(),
            self.project.outputDir(),
            self.project.outputDirHtml(),
            self.project.outputDirPdf(),
            self.project.outputDirHtmlStyles(),
            self.project.inputDirStyles(),
            self.project.outputDirEpub(),
            self.project.tempEpub(),
            self.project.tempScorm()]

    for directory in dirs:
        try:
            log.info('[PROYECTO] Creando el directorio "%s"', directory)
            utils.makeDirSafe(directory)
        except OSError as e:
            if e.errno == errno.EEXIST:
                raise HidraProjectAlreadyExistsException(e)
            raise
```

Anexo L: Código fuente de ProjectManager

Anexo M: Código fuente de Project

```
from . import strings
from .exceptions import HidraProjectNotLoadedException

import os
import yaml

import logging
log = logging.getLogger('hidra')


class Project(object):
    def __init__(self, projectDir):
        self.projectDir = projectDir
        self._yaml = None
        self._templatesDir = os.path.join(strings.Data.DATA,
                                          strings.Data.TEMPLATES)
        self._templatesImagesDir = os.path.join(self._templatesDir,
                                                strings.Data.IMAGES)

    @property
    def projectDir(self):
        return self._projectDir

    @projectDir.setter
    def projectDir(self, value):
        self._projectDir = value

    @property
    def templatesDir(self):
        return self._templatesDir

    @property
    def templatesImagesDir(self):
        return self._templatesImagesDir
```

```
def load(self):
    """Carga el archivo de proyecto en memoria"""

    path = os.path.join(self.projectDir, strings.Project.PROJECT_FILENAME)
    log.info('[PROYECTO] Abriendo el proyecto "%s"', path)
    with open(path, 'r') as hip_file:
        self._yaml = yaml.safe_load(hip_file)
        return self._yaml

def save(self):
    """Guarda los campos modificados al archivo de proyecto"""

    path = os.path.join(self.projectDir, strings.Project.PROJECT_FILENAME)
    log.info('[PROYECTO] Guardando el proyecto "%s" a disco', path)
    with open(path, 'w') as hip_file:
        yaml.dump(self._yaml, hip_file, default_flow_style=False)

# Getters/Setters de cada campo del archivo de proyecto
def tempEpub(self):
    return os.path.join(self.workDir(), 'epub')

def tempScorm(self):
    return os.path.join(self.workDir(), 'scorm')

def outputDirScorm(self):
    return os.path.join(self.outputDir(), strings.Project.OUTPUT_SCORM)

def outputDirEpub(self):
    return os.path.join(self.outputDir(), strings.Project.OUTPUT_EPUB)

def outputDirMobi(self):
    return os.path.join(self.outputDir(), strings.Project.OUTPUT_MOBI)

def outputDirHtml(self):
    return os.path.join(self.outputDir(), strings.Project.OUTPUT_HTML)

def outputDirPdf(self):
    return os.path.join(self.outputDir(), strings.Project.OUTPUT_PDF)

def outputDirHtmlStyles(self):
    return os.path.join(self.outputDir(), strings.Project.OUTPUT_HTML,
                        strings.Project.HTML_STYLES)

def outputDirHtmlImages(self):
    return os.path.join(self.outputDir(), strings.Project.OUTPUT_HTML,
                        strings.Project.HTML_IMAGES)
```

```
def inputDirStyles(self):
    return os.path.join(self._projectDir, strings.Project.STYLE)

def imagesDir(self, value=None, forcewrite=False):
    return os.path.join(self._projectDir,
                        self._field(strings.Project.GENERAL_SECTION,
                                    strings.Project.IMAGES_DIR,
                                    value, forcewrite))

def documentsDir(self, value=None, forcewrite=False):
    return os.path.join(self._projectDir,
                        self._field(strings.Project.GENERAL_SECTION,
                                    strings.Project.DOCUMENTS_DIR,
                                    value, forcewrite))

def workDir(self, value=None, forcewrite=False):
    return os.path.join(self._projectDir,
                        self._field(strings.Project.GENERAL_SECTION,
                                    strings.Project.WORK_DIR,
                                    value, forcewrite))

def outputDir(self, value=None, forcewrite=False):
    return os.path.join(self._projectDir,
                        self._field(strings.Project.GENERAL_SECTION,
                                    strings.Project.OUTPUT_DIR,
                                    value, forcewrite))

def pdfsize(self, value=None, forcewrite=False):
    return self._field(strings.Project.GENERAL_SECTION,
                       strings.Project.PDF_SIZE,
                       value, forcewrite)

def toclevel(self, value=None, forcewrite=False):
    return self._field(strings.Project.GENERAL_SECTION,
                       strings.Project.TOC_LEVEL,
                       value, forcewrite)

def style(self, value=None, forcewrite=False):
    return self._field(strings.Project.GENERAL_SECTION,
                       strings.Project.STYLE,
                       value, forcewrite)

def title(self, value=None, forcewrite=False):
    return self._field(strings.Project.PROJECT_SECTION,
                       strings.Project.TITLE,
                       value, forcewrite)
```

```
def author(self, value=None, forcewrite=False):
    return self._field(strings.Project.PROJECT_SECTION,
                       strings.Project.AUTHOR,
                       value, forcewrite)

def version(self, value=None, forcewrite=False):
    return self._field(strings.Project.PROJECT_SECTION,
                       strings.Project.VERSION,
                       value, forcewrite)

def cover(self, value=None, forcewrite=False):
    return os.path.join(self._projectDir,
                        self.imagesDir(),
                        self._field(strings.Project.PROJECT_SECTION,
                                    strings.Project.COVER,
                                    value, forcewrite))

def abstract(self, value=None, forcewrite=False):
    return os.path.join(self._projectDir,
                        self._field(strings.Project.PROJECT_SECTION,
                                    strings.Project.ABSTRACT,
                                    value, forcewrite))

def documents(self, value=None, forcewrite=False):
    return self._field(strings.Project.DOCUMENTS_SECTION,
                       strings.Project.LIST,
                       value, forcewrite)

def _field(self, section, field, value, forcewrite):
    """Permite leer o modificar un campo del archivo de proyecto"""

    if self._yaml is None:
        raise HidraProjectNotLoadedException

    # Escribirlo
    if value is not None:
        log.info('[PROYECTO] Guardando [%s][%s] = %s', section, field, value)
        self._yaml[section][field] = value
        if forcewrite:
            self.save()

    # Leerlo
    return self._yaml[section][field]
```

Anexo M: Código fuente de Project

Anexo N: Ejemplo de clase tag: TagImg

```
from .. import strings
from ..tag import Tag
from ..tags import tag_ref
from ..exceptions import HidraTemplateNotFoundException, HidraTagFormatException

import re
import os
from string import Template


class TagImg(Tag):
    def __init__(self, projectManager, args):
        super(self.__class__, self).__init__(projectManager)

        self.type = strings.Tags.TAGIMG
        self.tags = []
        self.isBlock = False
        for i in xrange(0, len(args[1])):
            s = args[1][i]
            if s[0] == strings.References.PATH:
                self.image = s[1]
                self.filepath = os.path.join(args[3], s[1])
            elif s[0] == strings.References.PERCENTAGE:
                self.percentage = s[1]
            elif s[0] == strings.References.DESCRIPTION:
                self.desc = s[1][1:-1]
            elif s[0] == strings.References.REFERENCE:
                self.idRef = s[1]
            elif s[0] == "nimg":
                self.nimg = s[1]
            else:
                if s[0][0] == strings.Tags.TAGREF:
                    key_options = s[0][1][0]
                    ref = tag_ref.TagRef(self.projectManager, key_options)
                    self.tags.append(ref)
```

```
    if not hasattr(self, 'filepath') or not hasattr(self, 'idRef') or not hasattr(self, '
        percentage') or not hasattr(self, 'desc'):
        raise HidraTagFormatException('Error en los argumentos de imagen en la seccion', self.
            nimg)

def getRef(self):
    return self.nimg

def getTag(self):
    cad = strings.Tags.HIDRA + ':' + self.type + ':' + self.identifier
    return cad

def processLatex(self):
    project = self.projectManager.project

    try:
        with open(os.path.join(project.templatesDir,
                               project.style(),
                               'latex_tag_img.hit')) as file:
            s = Template(file.read())
            if not self.tags:
                return s.substitute(width=str(float(self.percentage) / 100),
                                    path=self.filepath.replace('\\', '/'),
                                    desc=self.desc, idref=self.idRef, nimg=self.nimg)
            else:
                ref = self.tags[0]
                return s.substitute(width=str(float(self.percentage) / 100),
                                    path=self.filepath.replace('\\', '/'),
                                    desc=self.desc + ' ' + ref.processLatex(),
                                    idref=self.idRef, nimg=self.nimg)
    except EnvironmentError as e:
        raise HidraTemplateNotFoundException(e)

def processHtml(self):
    project = self.projectManager.project

    try:
        with open(os.path.join(project.templatesDir,
                               'html_tag_img.hit')) as file:
            s = Template(file.read())
            splitted = self.image.split('.')
            if splitted[1] == 'pdf':
                self.image = splitted[0] + '.jpg'
            if not self.tags:
                return s.substitute(width=self.percentage,
                                    path=os.path.join('images',
```

```
                                                self.image),
                                desc=self.desc, idref=self.idRef,
                                nimg=self.nimg)
            else:
                ref = self.tags[0]
                return s.substitute(width=self.percentage,
                                path=os.path.join('images',
                                                self.image),
                                desc=self.desc + ' ' +
                                ref.processHtml(),
                                idref=self.idRef, nimg=self.nimg)
    except EnvironmentError as e:
        raise HidraTemplateNotFoundException(e)
```

Anexo N: Ejemplo de clase tag: TagImg

Anexo O: Procesamiento SCORM

```
from .. import utils, strings
from ..hidra_task import HidraTask
from ..exceptions import HidraTemplateNotFoundException

import shutil
import re
import os
import operator
from zipfile import ZipFile
from string import Template
import logging
log = logging.getLogger('hidra')


class TaskProcessScorm(HidraTask):
    def __init__(self, projectManager):
        super(self.__class__, self).__init__(projectManager)
        self.cont = 0

    def run(self):
        """Tarea que genera el archivo 'output/scorm/salida.zip'"""

        project = self.projectManager.project

        # Leemos salida.html
        shutil.rmtree(project.tempScorm(), ignore_errors=True)
        shutil.rmtree(project.outputDirScorm(), ignore_errors=True)
        utils.makeDirSafe(project.tempScorm())
        salida_file_path = os.path.join(project.outputDirHtml(),
                                        'salida.html')

        # Separamos salida.html en varios archivos, tantos como capitulos tenga el libro
        header_regex = re.compile(strings.RegularExpresion.CHAPTER)
        ignore_block = False
        # Condicion y si hay que introducir un test usar la plantilla scorm_html_test_template.hit
```

```
with open(os.path.join(project.templatesDir,
                       'scorm_html_template.hit'), 'r') as in_template_file:
    in_template = in_template_file.read()
    with open(salida_file_path, 'r') as in_html:
        lines = in_html.readlines()
        for i, line in enumerate(lines):
            match = header_regex.match(line)
            if match is not None:
                chapter = match.group(1).replace('.', '_')

                with open(os.path.join(project.tempScorm(), 'file-{}.html'.format(chapter)),
                     'w') as out_html:
                    out_html.write(in_template)
                    if(chapter=='0_'):
                        # add line cover
                        out_html.write('<p><div class="hidra-cover"><img src="'+os.path.join
                            ('images', project.cover().replace('/', '\\').split('\\')[-1])+'
                            " /></div>')
                    if (chapter=='bib_'):
                        out_html.write('<h1 class="hidra-bib-title">Referencias</h1>')
                    for j in xrange(i, len(lines)):
                        if 'hidra-toc-begin' in lines[j]:
                            ignore_block = True
                        if 'hidra-toc-end' in lines[j]:
                            ignore_block = False

                        if ignore_block:
                            continue
                        if('<a href=\"#') in lines[j]:
                            label = re.findall(r'<a href=\"(#\w+\:\w+\:\w+)\">\[[0-9]+\]</a>
                                ', lines[j])
                            if label is not []:
                                for l in label:
                                    lines[j] = lines[j].replace(l, 'file-bib_.html'+l)

                        out_html.write(lines[j])

                        try:
                            match2 = header_regex.match(lines[j + 1])
                            if match2 is not None:
                                if (len(match2.group(1).split('.')) - 1) <= project.toclevel
                                    ():
                                    break
                        except IndexError:
                            pass

                    out_html.write('</body></html>')
```

```
    self.copyFilesScorm()
    files = os.listdir(project.tempScorm())
    manifest_path = os.path.join(project.tempScorm(), 'imsmanifest.xml')
    with open(manifest_path, 'a') as manifest:
        try:
            with open(os.path.join(project.templatesDir,
                                   'scorm_manifest.hit')) as file:
                s = Template(file.read())
                cad = s.substitute(TituloLibro=(project.title()).encode('utf-8'),
                                   items=self.metScorm(files),
                                   recursos=self.resourcesScorm())
                manifest.write(cad)
        except EnvironmentError as e:
            raise HidraTemplateNotFoundException(e)

    self.copyResourcesScorm()

    with ZipFile(os.path.join(project.workDir(), 'temp.zip'), 'w') as myzip:
        # Copiar archivos
        list = os.listdir(project.tempScorm())
        for l in list:
            if os.path.isdir(os.path.join(project.tempScorm(), l)):
                continue

            myzip.write(os.path.join(project.tempScorm(), l), l)

        # Copiar images
        directory = os.path.join(project.tempScorm(), 'images')
        for dirpath, dirs, files in os.walk(directory):
            mkdirectory = dirpath.split(os.sep)[-1]
            for f in files:
                myzip.write(os.path.join(directory, f), os.path.join(mkdirectory, f))

        # Copiar style
        directory = os.path.join(project.tempScorm(), 'styles')
        for dirpath, dirs, files in os.walk(directory):
            mkdirectory = dirpath.split(os.sep)[-1]
            for f in files:
                myzip.write(os.path.join(directory, f), os.path.join(mkdirectory, f))

    utils.makeDirSafe(project.outputDirScorm())
    shutil.copy(os.path.join(project.workDir(), 'temp.zip'),
                os.path.join(project.outputDirScorm(), 'salida.zip'))

def metScorm(self, files):
    lista = {}
```

```
    for i in xrange(len(files)):
        s = files[i].split('_')
        if('file-' in s[0]):
            lista[files[i]] = len(s) - 1
    sorted_list = sorted(lista.iteritems(), key=operator.itemgetter(0))
    self.cont = 0
    cad = self.itemsScorm(sorted_list, sorted_list[0][1])
    return cad

def itemsScorm(self, sorted_list, level):
    project = self.projectManager.project

    cad = ''
    title = ''
    while(self.cont < len(sorted_list)):
        x = self.cont
        with open(os.path.join(project.tempScorm(), sorted_list[self.cont][0]), 'r') as chapter:
            lines = chapter.readlines()
            for i, line in enumerate(lines):
                r = re.search(strings.RegularExpresion.H_HTML, line)
                if r is not None:
                    title = r.group(1)
                    break
                else:
                    title = 'Portada'
            try:
                with open(os.path.join(project.templatesDir,
                                       'scorm_items.hit')) as file:
                    s = Template(file.read())
                    t = '  ' * sorted_list[self.cont][1]
                    if(self.cont + 1 < len(sorted_list)):
                        if(sorted_list[self.cont + 1][1] > sorted_list[self.cont][1]):
                            self.cont += 1
                            cad = cad + s.substitute(Capitulo=sorted_list[self.cont - 1][0],
                                                     y=sorted_list[self.cont - 1][0],
                                                     TituloCap=title,
                                                     tab=t,
                                                     items=self.itemsScorm(sorted_list,
                                                          sorted_list[self.cont - 1][1]))
                        else:
                            cad = cad + s.substitute(Capitulo=sorted_list[self.cont][0],
                                                     y=sorted_list[self.cont][0],
                                                     TituloCap=title,
                                                     tab=t,
                                                     items='') + '      ' + t + '</item>\n'
                        if(sorted_list[x][1] != level and sorted_list[x + 1][1] < sorted_list[x
                             ][1]):
```

```
                            cad += '      ' + t + '</item>\n'
                    else:
                        cad = cad + s.substitute(Capitulo=sorted_list[self.cont][0],
                                                 y=sorted_list[self.cont][0],
                                                 TituloCap=title,
                                                 tab=t,
                                                 items='') + '      ' + t + '</item>\n'
            except EnvironmentError as e:
                raise HidraTemplateNotFoundException(e)
        self.cont += 1
    return cad

def resourcesScorm(self):
    project = self.projectManager.project

    cad = ''
    files = sorted(os.listdir(project.tempScorm()))
    for f in files:
        try:
            with open(os.path.join(project.templatesDir,
                                   'scorm_recursos.hit')) as file:
                s = Template(file.read())
                cad = cad + s.substitute(y=f,
                                         filename=f,
                                         rutas=self.rutasScorm(os.path.join(project.tempScorm(),
                                               f)))
        except EnvironmentError as e:
            raise HidraTemplateNotFoundException(e)
    return cad

def rutasScorm(self, f):
    project = self.projectManager.project

    cad = ''
    resources = []
    with open(f, 'r') as res:
        lines = res.readlines()
        for line in enumerate(lines):
            tokens = str(line).split('>')
            for i in range(len(tokens)):
                r = re.search(strings.RegularExpresion.PATH_RESOURCES, tokens[i])
                if r is not None:
                    if not (r.group(1) in resources):
                        resources.append(r.group(1))
    for cont in range(len(resources)):
        try:
            with open(os.path.join(project.templatesDir,
```

```
                                    'scorm_rutarecursos.hit')) as file:
                s = Template(file.read())
                cad = cad + s.substitute(rutarecursos=resources[cont])
        except EnvironmentError as e:
            raise HidraTemplateNotFoundException(e)
    return cad

def copyResourcesScorm(self):
    project = self.projectManager.project

    shutil.copytree(os.path.join(project.outputDirHtml(), 'images'), os.path.join(project.
        tempScorm(), 'images'))
    shutil.copytree(os.path.join(project.outputDirHtml(), 'styles'), os.path.join(project.
        tempScorm(), 'styles'))

def copyFilesScorm(self):
    project = self.projectManager.project

    files = os.listdir(os.path.join(project.templatesDir, 'scorm'))
    for f in files:
        shutil.copy(os.path.join(project.templatesDir, 'scorm', f), project.tempScorm())
```

Anexo O: Procesamiento SCORM

Anexo P: Ejemplo cambio de estilos

Chapter 1

Introducción

En los últimos siglos han ocurrido importantes avancez tecnológicos que han propiciado enormes cambios en el modo de vida del ser humano. La publicación no es una excepción. Si bien es cierto que el hito más importante en este campo fue gracias a la invención de la imprenta moderna a mediados del siglo XV, en los últimos años han sucedido avances tecnológicos muy significativos en el ámbito de la edición digital.

Las ventajas que han proporcionado estos sistemas son cuantiosas, pero también presentan algunas dificultades. El que nos concierne para el presente proyecto es la heterogeneidad de formatos, lo que puede suponer que la tarea de obtener un mismo documento en distintos formatos sea tediosa y molesta para el usuario.

El proyecto HIDRA surge de la necesidad de crear un sistema que facilite la producción de archivos a distintos formatos. El objetivo es la definición de un lenguaje intuitivo para la definición de los distintos elementos que se deseen añadir a la publicación, y universal para todos los formatos de salida que contemple el sistema.

Sistemas de publicación

La historia no se entiende sin los registros por escrito de de todas las épocas. En la prehistoria, los hombres dejaban sus inscripciones talladas en la roca, si bien, la historia de la escritura no se iniciaría hasta miles de años después, estos constituyen los primeros antecedentes de la escritura.

En la Mesopotamia del IV milenio a.C. se encuentran los orígenes del que durante siglos ha sido la principal fuente de transmisión de información, el libro. En su origen, los primeros soportes fueron las tablillas de arcilla de los sumerios, haciendo uso de la escritura pictogramática, la cual también era usada por los

Capítulo 1

Introducción

En los últimos siglos han ocurrido importantes avancez tecnológicos que han propiciado enormes cambios en el modo de vida del ser humano. La publicación no es una excepción. Si bien es cierto que el hito más importante en este campo fue gracias a la invención de la imprenta moderna a mediados del siglo XV, en los últimos años han sucedido avances tecnológicos muy significativos en el ámbito de la edición digital.

Las ventajas que han proporcionado estos sistemas son cuantiosas, pero también presentan algunas dificultades. El que nos concierne para el presente proyecto es la heterogeneidad de formatos, lo que puede suponer que la tarea de obtener un mismo documento en distintos formatos sea tediosa y molesta para el usuario.

El proyecto HIDRA surge de la necesidad de crear un sistema que facilite la producción de archivos a distintos formatos. El objetivo es la definición de un lenguaje intuitivo para la definición de los distintos elementos que se deseen añadir a la publicación, y universal para todos los formatos de salida que contemple el sistema.

Sistemas de publicación

La historia no se entiende sin los registros por escrito de de todas las épocas. En la prehistoria, los hombres dejaban sus inscripciones talladas en la roca, si bien, la historia de la escritura no se iniciaría hasta miles de años después, estos constituyen los primeros antecedentes de la escritura.

En la Mesopotamia del IV milenio a.C. se encuentran los orígenes del que durante siglos ha sido la principal fuente de transmisión de información, el libro. En su origen, los primeros soportes fueron las tablillas de arcilla de los sumerios, haciendo uso de la escritura pictogramática, la cual también era usada por los egipcios, que la desarrollaron hasta usar la escritura a palabras y los rollos de papiro.

Tras el papiro llega el pergamino, aunque ambos conviven en Roma durante cientos de años. Si bien es cierto, que el formato de rollo fue dejando lugar a los *codex*, con un formato rectangular que recuerdan a los que actualmente conocemos como *libro*.

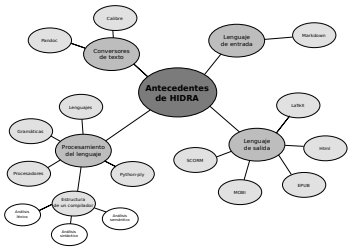


Imagen 3.1: Mapa conceptual de los antecedentes

ha decidido hacer uso de algunas de las funcionalidades que proporciona, estás están explicadas en el Módulo de conversión de documentos.

A continuación se muestra un ejemplo de uso de pandoc:

```
1 pandoc file.html --highlight-style=pygments --webtex file.epub
```

Código 3.1: Prueba de uso de Pandoc

En el ejemplo, la opción *–highlight-style=pygments* indica el estilo de color que se usa para resaltar la sintaxis, y *–webtex* renderiza las fórmulas haciendo uso de un script externo a Pandoc.

[http://hackage.haskell.org/package/pandoc]

> ⚠ Puede convertir documentos en markdown, reStructuredText, textile, HTML, DocBook, LaTeX, MediaWiki markup, TWiki markup, OPML, Emacs Org-Mode, Txt2Tags, Microsoft Word docx, LibreOffice ODT, EPUB, o Haddock markup a HTML, docs, odt, xml, EPUB, DocBook, TEI Simple, GNU TexInfo, Groff man pages, Haddock markup, InDesign ICML, OPML, LaTeX, ConTeXt, PDF, Markdown, reStructuredText, AsciiDoc, MediaWiki markup, DokuWiki markup, Emacs Org-Mode y Textile.

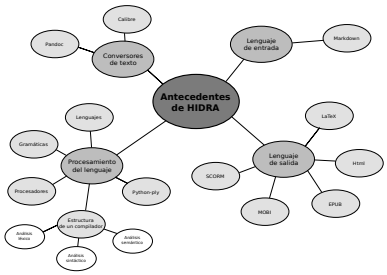


Figura 3.1: Mapa conceptual de los antecedentes

Conversores de texto

En esta sección se repasan distintas herramientas que cumplen algunos de los objetivos de HIDRA, con algunas de las funcionalidades que se desean para el sistema.

Pandoc

[http://pandoc.org/] Pandoc es un conversor universal de documentos, además es libre y de código abierto. Fue desarrollado por John MacFarlane, profesor de filosofía en la Universidad de California. Además cuenta con un módulo, denominado *Pandoc-citeproc* que se encarga de la gestión de referencias bibliográficas. Además, es una herramienta multiplataforma y fácil de usar. Para HIDRA se ha decidido hacer uso de algunas de las funcionalidades que proporciona, estás están explicadas en el Módulo de conversión de documentos.

A continuación se muestra un ejemplo de uso de pandoc:

```
1 pandoc file.html --highlight-style=pygments --webtex file.epub
```

En el ejemplo, la opción *–highlight-style=pygments* indica el estilo de color que se usa para resaltar la sintaxis, y *–webtex* renderiza las fórmulas haciendo uso de un script externo a Pandoc.

[http://hackage.haskell.org/package/pandoc]

$$G = (\Sigma_N, \Sigma_T, S, P) \tag{3.1}$$

será notado como *L(G)*, y se define como el conjunto de cadenas formadas por símbolos terminales que son derivables a partir del símbolo inicial de la gramática.

$$L(G) = \{u \varepsilon \Sigma_T | S^* \Rightarrow u\} \tag{3.2}$$

Chomsky realizó una clasificación tanto de las gramáticas como de los lenguajes formales, dividiéndolos en cuatro tipos, siendo L el lenguaje definido por la gramática G:

- *Lenguajes regulares: (Tipo 3)* Los lenguajes regulares pueden ser definidos por un autómata finito, y se utilizan para definir el léxico de los lenguajes de programación.
- *Lenguajes independientes de contexto: (Tipo 2)* Son los generados por las gramáticas libres o independientes de contexto.
- *Lenguajes sensibles (o dependientes) al contexto: (Tipo 1)* Las gramáticas de Tipo 1 generan este tipo de lenguajes. Ver la sección Gramáticas formales para más detalles.
- *Lenguajes sin restricciones: (Tipo 0)* La gramática de L (o gramáticas) no cumple ninguno de los casos anteriores.

La relación que existe entre los lenguajes que acaban de definirse es la que se muestra en la figura 3.5:

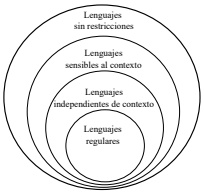


Imagen 3.5: Clasificación de los lenguajes de Chomsky

Formalmente, y con respecto a la definición formal de gramática que se muestra en la siguiente subsección Gramáticas formales, el lenguaje generado por una gramática:

$$G = (\Sigma_N, \Sigma_T, S, P) \tag{3.1}$$

será notado como *L(G)*, y se define como el conjunto de cadenas formadas por símbolos terminales que son derivables a partir del símbolo inicial de la gramática.

$$L(G) = \{u \varepsilon \Sigma_T | S^* \Rightarrow u\} \tag{3.2}$$

Chomsky realizó una clasificación tanto de las gramáticas como de los lenguajes formales, dividiéndolos en cuatro tipos, siendo L el lenguaje definido por la gramática G:

- *Lenguajes regulares: (Tipo 3)* Los lenguajes regulares pueden ser definidos por un autómata finito, y se utilizan para definir el léxico de los lenguajes de programación.
- *Lenguajes independientes de contexto: (Tipo 2)* Son los generados por las gramáticas libres o independientes de contexto.
- *Lenguajes sensibles (o dependientes) al contexto: (Tipo 1)* Las gramáticas de Tipo 1 generan este tipo de lenguajes. Ver la sección Gramáticas formales para más detalles.
- *Lenguajes sin restricciones: (Tipo 0)* La gramática de L (o gramáticas) no cumple ninguno de los casos anteriores.

La relación que existe entre los lenguajes que acaban de definirse es la que se muestra en la figura 3.5:

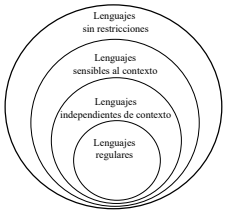


Figura 3.5: Clasificación de los lenguajes de Chomsky

Siguiendo con la definición del lenguaje, son necesarios tres aspectos fundamentales:

- **Léxico:** Constituido por el conjunto de palabras que pertenecen al lenguaje, y sus respectivas categorías.

En el *módulo de preprocesamiento* se describe qué y cómo se obtienen los atributos que se utilizan en los constructores de las etiquetas, ver para más precisión. El *módulo de etiquetas* cuenta con una clase principal, **Tag** de la que heredan las etiquetas más específicas. Sus atributos son un identificador único que se genera automáticamente en el constructor de la clase, el *projectManager* y una variable que indica si la etiqueta es de bloque o no; además de unos métodos para obtener o modificar el contenido de las variables mencionadas, y los métodos más importantes: **processLatex** y **processHtml**.

Cada clase en su constructor contiene unos atributos que dependen del tipo de etiqueta que sean, por ejemplo, en el caso de las imágenes algunos de sus atributos son **filepath**, que almacena la ruta de la imagen, **percentaje** o **desc**, que son el ancho de la imágen y la descripción de la etiqueta respectivamente; en cambio, la de código tiene **lang**, que contiene el lenguaje del código; el constructor se encarga de dar el valor correcto de estos atributos, obtenidos del *módulo de preprocesamiento*, se muestra un ejemplo en el siguiente código:

```

def __init__(self, projectManager, args):
    super(self.__class__, self).__init__(projectManager)

    self.type = strings.Tags.TAGCODEBEGIN
    self.tags = []
    self.isBlock = True
    for i in xrange(0, len(args[1])):
        s = args[1][i]
        if s[0] == strings.References.LANGUAGE:
            self.lang = s[1]
        elif s[0] == strings.References.DESCRIPTION:
            self.desc = s[1][1:-1]
        elif s[0] == strings.References.REFERENCE:
            self.idRef = s[1]
        elif s[0] == 'ncode':
            self.ncode = s[1]
        else:
            if s[0][0] == strings.Tags.TAGREF:
                key_options = s[0][1][0]
                ref = tag_ref.TagRef(self.projectManager,
                    key_options)
                self.tags.append(ref)
```

Código 6.3: Constructor TagCodeBegin

Una vez almacenados los atributos, y dependiendo del procesado que sea necesario se invocan los métodos **processLatex** y **processHtml**. Cada formato cuenta con una serie de plantillas para cada etiqueta, además de tener definidos los

MÓDULO DE ETIQUETAS 49

```

def __init__(self, projectManager, args):
    super(self.__class__, self).__init__(projectManager)

    self.type = strings.Tags.TAGCODEBEGIN
    self.tags = []
    self.isBlock = True
    for i in xrange(0, len(args[1])):
        s = args[1][i]
        if s[0] == strings.References.LANGUAGE:
            self.lang = s[1]
        elif s[0] == strings.References.DESCRIPTION:
            self.desc = s[1][1:-1]
        elif s[0] == strings.References.REFERENCE:
            self.idRef = s[1]
        elif s[0] == 'ncode':
            self.ncode = s[1]
        else:
            if s[0][0] == strings.Tags.TAGREF:
                key_options = s[0][1][0]
                ref = tag_ref.TagRef(self.projectManager, key_options)
                self.tags.append(ref)
```

Una vez almacenados los atributos, y dependiendo del procesado que sea necesario se invocan los métodos **processLatex** y **processHtml**. Cada formato cuenta con una serie de plantillas para cada etiqueta, además de tener definidos los estilos para cada uno de ellos previamente, en estos métodos se lee el fichero que contiene la plantilla que corresponda, y sustituye las variables por el valor que tiene. Por ejemplo, continuando con el ejemplo de código, para latex, el contenido de la plantilla sería:

$$begin\{code\}\{\$lang\}\{\$ncode : \$desc\}\{\$idref\}$$

donde las variables se identifican por el símbolo que las precede. Y su sustitución en el método *processLatex*:

```


s = Template(file.read())
return s.substitute(idref=self.idRef, lang=self.lang,
                    ncode=self.ncode, desc=self.desc)
```

Para html el proceso es el mismo, y para el resto de formatos que no se han mencionado, como ePub, azw o SCORM, al derivar de html, es suficiente con la versión en *html*.

Gestión de imágenes

La gestión de las imágenes, es una tarea muy especializada en los distintos formatos de salida del sistema, ya que cada uno de ellos tiene necesidades diferentes, y con HIDRA quedan satisfechas, por ejemplo, para el formato html, haciendo uso de **ImageMagick** las imágenes son procesadas para obtener una versión más ligera de las mismas, en formato *.jpg*. Para latex, si es posible, el sistema mantendrá la versión vectorial, o en su defecto las procesará también.

```
7                os.path.join(project.workDir(), 'temp.epub'))
8 Unzip().run(os.path.join(project.workDir(), 'temp.epub'),
      project.tempEpub())
```

Código 6.16: Conversión a .epub

A continuación hay que realizar distintas tareas, por un lado, concatenar todos los estilos en un único CSS, en segundo lugar, obtener la lista de XHTMLs generados por Pandoc y añadirles los estilos. Por otro lado las imágenes deben añadirse al archivo epub, y para ello hay que prepararlas, por medio de **Imagemagick**:

```
1
2
3 Imagemagick().resizeForEpub(os.path.join(project.tempEpub(),
      strings.Project.HTML_IMAGES, img))
```

Código 6.17: Preparación de imágenes para ePub

El siguiente paso sería añadir las rutas de las imágenes al "content.opf", autor y titulo.

Y por último comprimir el ePub ya procesado. Para ello se copian los archivos generados, los media y META-INF.

Procesado Azw

Para el procesado MOBI o azw, que es una variante del primero, se hace uso de **Calibre** y simplemente se convierte, sin realizar cambios posteriormente.

```
1
2
3 Calibre().run(os.path.join(project.workDir(), 'temp.epub'),
4                       os.path.join(project.outputDirMobi(),
                            'salida.azw3'),
5                       os.path.join(project.tempEpub(),
                            'content.opf'))
```

Código 6.18: Conversión a .azw

Procesado SCORM

Para comenzar el procesado SCORM, lo primero es separar *salida.html*, que es el resultado del procesado html, en varios archivos, tantos como capítulos tenga el libro, o secciones en las que se divida el proyecto. Cada uno de estos capítulos necesita algunas modificaciones, como expresiones sobrantes o caracteres que hay que cambiar, pero la dificultad en este procesado consiste en la edición del *imsmanifest.xml*.

El siguiente paso sería añadir las rutas de las imágenes al "content.opf", autor y titulo.

Y por último comprimir el ePub ya procesado. Para ello se copian los archivos generados, los media y META-INF.

Procesado Azw

Para el procesado MOBI o azw, que es una variante del primero, se hace uso de **Calibre** y simplemente se convierte, sin realizar cambios posteriormente.

```
1
2
3 Calibre().run(os.path.join(project.workDir(), 'temp.epub'),
4                       os.path.join(project.outputDirMobi(), 'salida.azw3'),
5                       os.path.join(project.tempEpub(), 'content.opf'))
```

Procesado SCORM

Para comenzar el procesado SCORM, lo primero es separar *salida.html*, que es el resultado del procesado html, en varios archivos, tantos como capítulos tenga el libro, o secciones en las que se divida el proyecto. Cada uno de estos capítulos necesita algunas modificaciones, como expresiones sobrantes o caracteres que hay que cambiar, pero la dificultad en este procesado consiste en la edición del *imsmanifest.xml*.

La estructura que debe tener este documento es muy concreta, y distinta en cada proyecto, la idea se basa en la división inicial del archivo *salida.html*, dependiendo de su posición en el proyecto, a los distintos capítulos se le asigna un nombre con una estructura determinada:

file-capitulo_subcapitulo.html

donde capítulo y subcapítulo se sustituirán por los números correspondientes. A la hora de generar el archivo *imsmanifest.xml* se basa en estos nombres, listando los archivos y añadiéndolos donde corresponde. Además, es necesario conocer los archivos media que contiene cada capítulo, para cambiar las rutas y añadirlos también.

El proceso completo puede verse en el anexo O.

Submódulo de keywords

Se ha mencionado a lo largo del presente documento, pero las *keywords* son imprescindibles en el proyecto para algunas tareas como la inserción de una portada, la tabla de contenido o la bibliografía.

En el archivo de proyecto, **Projecto.hip** se añaden las keyword que se deseen, por ejemplo, para la tabla de contenidos y la bibliografía basta con añadir:

Bibliografía

[1] AHO A.V. , R. SETHI Y J.D. ULLMAN *Compilers: Principles, techniques, and tools*, Addison-Wesley, 1986.

[2] AHO A.V., J.D. ULLMAN *Principles of Compiler Design*, Addison-Wesley, 1977.

[3] ADVANCED DISTRIBUTED LEARNING *Sharable Content Object Reference Model Version 1.2 The SCORM Overview*, 2001.

[4] AMO, F.A. NORMAND, L.M. PÉREZ, F.J.S. *Introducción a la ingeniería del software*, Delta Publicaciones, España, 2005.

[5] CHAN, J. *Learn Python in One Day and Learn It Well: Python for Beginners with Hands-on Project. The only book you need to start coding in Python immediately*, Createspace, Estados Unidos, 2015.

[6] EPUB 3 OVERVIEW RECOMMENDED SPECIFICATION *http://www.idpf.org/epub/301/spec/epub-overview.html*, 26 June 2014.

[7] INTECO *Ingeniería del Software: Metodologías y ciclos de vida*, Ministerio de Industria, Turismo y Comercio, España, 2009.

[8] LAMPORT, L. *LATEX: A Document Preparation System*, Addison-Wesley, segunda edición, 1994.

[9] M. GOOSSENS, F. MITTELBACH, Y A. SAMARIN *The LATEX Companion*, Addison-Wesley, 1994.

[10] STEFANOV, S. *JavaScript Patterns*, O'Reilly Media, 2010.

[11] TANEMBAUM, A. *Compilers: Principles, Tecniques, and Tools*, Tomo 1. ACM Press. 5ta Edición.

[12] PANDOC *http://pandoc.org/*.

[13] PANDOC: CONVERSION BETWEEN MARKUP FORMATS *http://hackage.haskell.org/package/pandoc*.

[14] CALIBRE *https://calibre-ebook.com*.

[15] PYTHON-PLY http://www.dabeaz.com/ply/.

[16] MARKDOWN http://markdown.es/.

[17] LATEX https://www.latex-project.org/.

[18] INTERNATION DIGITAL PUBLISHING FORUM: EPUB http://idpf.org/epub.

[19] HTML https://www.w3.org/html/.

[20] THE UNICODE CONSORTIUM http://www.unicode.org/.

[21] MOBIPOCKET http://www.mobipocket.com/en/DownloadSoft/ProductDetailsCreator.asp.

[22] SHARABLE CONTENT OBJECT REFERENCE MODEL https://docs.moodle.org/all/es/SCORM.

[23] KINDLE https://kindle.amazon.com/.

[24] MICROSOFT WINDOWS http://www.microsoft.com.

[25] GNU LINUX http://www.gnu.org/gnu/linux-and-gnu.es.html.

[26] GIT http://git-scm.com.

[27] TRELLO https://trello.com/.

[28] MOODLEL https://docs.moodle.org.

[29] PYTHON https://www.python.org.

[30] JAVASCRIPT https://www.javascript.com/.

[31] THE GTK+ PROJECT http://www.gtk.org/.

[32] IMAGEMAGICK http://www.imagemagick.org.

[33] BIBTEX2HTML https://www.lri.fr/ filliatr/bibtex2html/.

[34] TEX2IM http://www.nought.de/tex2im.php.

[35] RELOAD: REUSABLE ELEARNING OBJECT AUTHORING DELIVERY http://www.reload.ac.uk/.