UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

Visualizador de documentos con editor de anotaciones para PC y Mac con la tecnología JavaFX

CARLOS HERNANDO MONFORTE

2014

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

VISUALIZADOR DE DOCUMENTOS CON EDITOR DE ANOTACIONES PARA PC Y MAC CON LA TECNOLOGÍA JAVAFX

Autor: Carlos Hernando Monforte
Director: Antonio Moratilla Ocaña

TRIBUNAL:

Presidente:.....

Vocal 1º:.....

Vocal 2º:.....

CALIFICACIÓN:.....

FECHA:.....

DEDICATORIA

Quiero agradecer a mi novia todas las horas que me ha compartido con la universidad.

A mis padres y hermanos por darme apoyo en estos años de tan duro trabajo educativo y laboral.

A todos los profesores que me han formado en mi trayectoria académica, en especial a Luis Uriarte que fue quien me inició en el mundo de la informática.

RESUMEN

Con este trabajo quiero mostrar el resultado del desarrollo de una aplicación con la cual podremos visualizar documentos y editar todo tipo de anotaciones sobre el mismo.

Para ello primero analizaré las diferentes tecnologías que actualmente se pueden usar para el desarrollo de aplicaciones de escritorio y las compararé entre sí encontrando la que mejor se adapta a nuestras necesidades.

También explicaré el formato de entrada y salida a la aplicación así como la estructura diseñada para el desarrollo de la aplicación, analizando los problemas que han ido surgiendo a lo largo del proceso para terminar con una planificación y presupuesto necesario para realizar el proyecto.

La finalidad de este proyecto es aprender a desarrollar aplicaciones de escritorio; siguiendo una metodología y con las herramientas de diseño que hemos aprendido a lo largo de estos años de estudio, consiguiendo un código más limpio, usable y fácil de extender.

SUMMARY

With this work I want to show the development of an application with which we can view documents and edit all kinds of annotations on it.

For this first I will discuss the different technologies currently be used for the development of desktop applications and compare it to each other by finding the best suited to our needs.

Also explain the format of input and output to the application and the structure designed for application development, examining problems that have arisen throughout the process to end planning and budget needed for the project.

The purpose of this project is to learn how to develop desktop applications; following a methodology and design tools that we have learned over the years of study, getting a clean, usable code and easy to extend.

PALABRAS CLAVE

JavaFX, RIAs, NetBeans, Visor de documentos, editor de anotaciones, JSON, Patrones de diseño,

Tabla de contenido

TABLA ILUSTRACIONES

Parte I: Introducción

1. Introducción

La última asignatura a la que nos enfrentamos en el grado o ingeniería es el proyecto final de grado (en adelante PFG) con el cual completamos el total de los créditos necesarios para graduarnos mediante la realización de un trabajo de investigación, desarrollo o de ambos, sobre alguna tecnología de última generación, con el cual el estudiante debe poner en práctica todos los conocimientos que ha adquirido a lo largo de sus años de estudio.

La informática es una ciencia muy amplia y multidisciplinar. Aprender y poder abarcar todos los conceptos o conocer y desarrollar en todas las tecnologías es una aventura titánica. Por esto lo que buscará cualquier estudiante con el PFG es un reto con el que seguir aprendiendo y profundizando en el conocimiento de esta ciencia en una de las múltiples ramas y disciplinas de las que hay.

Como alumno de una carrera técnica informática me atrae la parte del desarrollo más que la investigación. Y de entre todas las ramas de la informática siempre sentí cierta predilección por el arte de la programación, por lo que decidí enfocar mi PFG al desarrollo de una aplicación con una tecnología muy interesante.

A pesar de trabajar desde hace algunos años en el mundo de la programación dentro de las telecomunicaciones en el desarrollo de sistemas empotrados, siempre me he dedicado al mismo enfoque y encontré un reto, al que nunca me había enfrentado, en la programación de una aplicación de escritorio con interfaces de usuario (UI). Así que busqué de qué tecnología podría tratar mi PFG.

Hablando con un profesor me sugirió una idea que él tenía que se basaba en ese tipo de programación a la que me quería enfrentar, lo cual me serviría para agrandar mi experiencia y profundizar en el conocimiento de los distintos tipos de programación que existen.

La aplicación que quería es un lector de documentos en el cual se podrán realizar anotaciones de texto, dibujos, subrayados y todo tipo de anotaciones que se pudiesen imaginar. Pero el lector tendría algunas particularidades como que el fichero del documento y el de las anotaciones no será el mismo, o que la aplicación tendría animaciones que le darían mayor vistosidad, etc. Este tipo de particularidades es lo que hará que la aplicación sea distinta con respecto a lo que nos ofrece el mercado actualmente.

Como no sabía mucho sobre este tema empecé leyendo sobre cómo se desarrollan las aplicaciones de escritorio, y ante mi inexperiencia me di cuenta que todo lo aprendido en mis años de trabajo poco servirían para diseñar y desarrollar este tipo de aplicación. Me propuse informarme sobre cómo encarar este nuevo proyecto. Lo primero que leí fue sobre patrones de diseño, los cuales según avanzó el desarrollo me di cuenta de la importancia de dominarlos. También tuve que acostumbrarme a una nueva tecnología como es JavaFX, de la cual tiene diferencias, que más tarde describiré, con SWING o AWT que son las que usé durante las asignaturas que me requerían programar con el lenguaje Java.

Con este trabajo quiero mostrar todas las fases de desarrollo de software con esta tecnología. El propósito de la memoria es recoger mi experiencia en el desarrollo de la aplicación para que sirva como documentación de base para otros estudiantes, descubran el potencial de esta tecnología y adquieran unos conocimientos básicos que le permitan iniciarse en el desarrollo de aplicaciones con JavaFX.

2. Objetivos

Cuando empezamos este PFG teníamos una serie de objetivos en mente que teníamos que cumplir. A continuación detallaremos los objetivos:

Como objetivo principal decidimos **el diseño y desarrollo de una aplicación de escritorio para Windows y Mac OS X, con una estética atractiva para el usuario y que fuera funcione de manera distinta al resto de las aplicaciones existentes en el mercado hasta la fecha.**

Para ayudar a cumplir este gran objetivo planeamos los siguientes subobjetivos:

- **Estudio de la tecnología de JavaFX.** Para poder desarrollar una aplicación con la tecnología que ofrece JavaFX primero debemos conocerla lo suficiente como para saber cuáles son las novedades que esta tecnología ofrece. Para ello se hará una búsqueda bibliográfica profunda que sirva como base para conocer JavaFX y poder usarlos en la aplicación. También se realizarán diferentes pruebas para poder familiarizarse con la sintaxis y características que la tecnología ofrece.

- **Comprensión y uso de patrones de diseño software.** Todo buen programador debe tener unas herramientas con las que poder diseñar y desarrollar sus aplicaciones. Con los patrones de diseño tenemos soluciones a problemas que se presentan con asiduidad. De forma que usando estas herramientas adaptadas a nuestras necesidades se conseguirá un software más robusto, fácil de mejorar y de extender.

- **Estudio y uso de formato jSON.** Los ficheros jSON son un formato de datos muy extendidos en la actualidad para todo tipo de aplicaciones debido a sus ventajas sobre otros formatos se va a usar para las comunicaciones de la aplicación, por lo cual primero habrá que entenderlo correctamente para después usarlo en la aplicación de forma correcta.

- **Desarrollo de aplicación con JavaFX.** Una vez estudiados y comprendidos todas las herramientas, tecnologías y formatos que eran necesarios se debían plasmar en una aplicación que iba a ser el objeto central del PFG.

- **Redactar un informe o memoria final**. La redacción de la memoria que está leyendo en estos instantes es la manera se tiene de plasmar todo el trabajo realizado y de documentar la aplicación que se ha desarrollado.

3. Estado del arte

3.1. RIAs

Hace algunos años, las aplicaciones informáticas se ejecutaban en un ordenador central desde el que los usuarios no podían interactuar con la información que contenían los servidores centrales debido a los elevados costes de procesamiento y transmisión de datos. Con la evolución de los ordenadores y su aumento de capacidades nació la arquitectura cliente/servidor, que permite repartir la capacidad de proceso entre los equipos de cliente y el servidor central.

Con este avance mejoró la escalabilidad y la centralización de la gestión de la información, aunque presenta algunas desventajas, como la congestión del tráfico de datos y su elevado coste, ya que es necesario desarrollar un software de cliente y de servidor.

En los años noventa hubo un importante auge en el uso de Internet, surgiendo el modelo de aplicaciones informáticas basadas en los navegadores. Su funcionamiento estaba basado en el lenguaje HTML y en el envío de peticiones a un servidor de aplicaciones, el cual se encargaba de escribir las páginas de forma dinámica y enviárselas al cliente. Pero este modelo presentaba un gran problema, la continua recarga de páginas por cada mínimo cambio provocaba un alto tráfico en la web.

En la actualidad se ha tratado de dar solución a los problemas de las aplicaciones web combinando las ventajas que ofrecen las aplicaciones Web y las aplicaciones tradicionales, naciendo la tecnología RIA como una nueva generación de aplicaciones que, sin duda, marcarán el futuro de los sistemas de información de empresas y corporaciones.

Las aplicaciones RIA pueden utilizar, al igual que las aplicaciones tradicionales de Internet, un navegador Web, pero cargan desde el principio toda la aplicación en el cliente, y sólo se produce comunicación con el servidor cuando se necesitan datos externos, ya sean de base de datos o de ficheros adicionales. Su arquitectura se basa en una aplicación-cliente y una capa de servicios separada, pareciéndose más en este aspecto a las antiguas aplicaciones cliente-servidor, con la diferencia de que únicamente solicitan datos del servidor, no necesitando

ninguna otra información. Además son capaces de trabajar de forma asíncrona y sin conexión con el servidor, lo que proporciona una versatilidad inmejorable ante problemas de conexión.

Uno de los beneficios de las aplicaciones RIA es la importante mejora en la experiencia visual del usuario, haciendo de la aplicación una experiencia para el usuario muy sencilla de usar, además de ofrecer mejoras en la conectividad, despliegue instantáneo de la aplicación, agilidad de acceso y garantía de desvinculación de la capa de presentación, permitiendo el uso de la aplicación desde cualquier ordenador conectado a Internet y sobre cualquier sistema operativo.

Son muchas las herramientas disponibles para la creación de entornos RIA, ofrecidas por distintos fabricantes y que, con unos costes razonables, permiten el desarrollo de este tipo de aplicaciones con presupuestos aceptables.

3.1.1 Tecnologías disponibles

Hay muchas tecnologías para el desarrollo de aplicaciones RIA pero entre todas estas tecnologías las más importantes son Adobe Flash, Microsoft Silverlight, y JavaFX por su penetración en el mercado del 96%, 66%, y el 76% respectivamente (reporte de agosto de 2012).

Adobe Flash

- ***Historia***

Adobe Flash tiene un origen en una aplicación, desarrollada por Jonathan Gay, llamada SmartSketch. Esta aplicación era para poder dibujar con un lápiz digital en ordenadores que tenían el sistema operativo PenPoint OS. Como PenPoint OS no tuvo mucho éxito la aplicación fue exportada a los sistemas operativos más utilizados Microsoft Windows y Mac OS.

Con el auge de internet, FutureWave, la empresa propietaria de la aplicación, fue añadiendo funcionalidades de dibujo vectorial y lanzó FutureSplash un programa de animación multiplataforma que fue usado por Microsoft y Disney Online para algunos trabajos en internet.

En 1996 Macromedia compró FutureSplash y lanzó Flash que comenzó como una herramienta de animación y un complemento opcional para los navegadores

En 1997, salió al mercado Macromedia Flash 2. Sus principales características fueron el uso de vectores en movimiento y algunos gráficos rasterizados. También se incluyó el sonido en estéreo de forma limitada, además de mejorar la integración de mapas de bits, botones, la Biblioteca y la capacidad de interpolar cambios de color.

Un año más tarde, en 1998, se comercializaría la versión 3 de Macromedia Flash cuyas mejoras con respecto a la anterior fueron directamente a la animación, reproducción y publicación, así como la introducción de comandos sencillos para obtener interactividad. Se añadieron la transparencia alfa y la licencia de compresión MP3.

En Mayo del 1999 apareció una nueva versión de Macromedia Flash. En esta cuarta versión hubo una mejora sustancial con respecto a la anterior versión al añadir el streaming de MP3 y la interpolación de movimiento. En esta versión se incluye el manejo de variables y comandos llamados "ActionScript".

La versión numero 5 de Macromedia Flash, fue lanzada en el año 2000 y supuso un gran paso adelante en sus características con la evolución de las capacidades de scripting de Flash (ActionScript). Macromedia también agregó la opción de personalizar la interfaz del entorno de edición.

En 2005, Adobe adquirió Macromedia y se hizo cargo del desarrollo de la tecnología Flash.

Flash 8 fue lanzado en 2005 y permitía video con canal alfa (On2 VP6). También permitía filtros, blends, simulador de dispositivos móviles, suavizado de textos y otras características.

En el año 2007 Adobe incorporó Flash a su entorno profesional llamado CS3. Este entorno estaba disponible para Microsoft Windows y para Mac. Adobe Flash CS3 permitía crear contenido interactivo de alta calidad para plataformas móviles, digitales y web.

Con las versiones de Adobe Flash CS4 y CS5 además de mejorar sus características como la animación, el cambio de pinceles, etc. Adobe intentó acercar una aplicación compleja a todo tipo de público ya fuera profesional o no profesional. Para ello simplificó la apariencia y organización de los menús y se añadió nuevas funciones para hacer su manejo más asequible.

Adobe Flash CS6 fue la última gran versión de esta tecnología ya que Apple y Microsoft, con Steve Jobs y Steve Ballmer a la cabeza, decidieron acabar con el reinado de Flash. Al menos en el sentido de que ambas compañías comenzaron a utilizar otras tecnologías diferentes en sus compañías.

- ***Características***

Disponibilidad en Sistemas Operativos de escritorio. Adobe asegura el funcionamiento de Flash en muchos sistemas operativos, especialmente los más importantes como son Windows, Mac OS X y Linux.

Funciona en arquitecturas de 64-bit. Desde la versión 11 de Adobe Flash, lanzada en octubre de 2011, se puede usar la aplicación en PC con arquitectura de 32-bit y en las que son de 64.bit.

Disponibilidad en Sistemas Operativos móviles. Existe Adobe Flash para una gran variedad de Sistemas Operativos móviles como pueden ser Android (desde la versión 2.2) Pocket PC/Windows CE, Symbian, Palm OS y webOS (desde la versión 2.0). El único Sistema Operativo que nunca ha soportado Adobe Flash ha sido iOS, debido a la negativa del presidente de Apple Steve Jobs.

Disponibilidad en otros dispositivos. Adobe Flash Lite es una versión reducida de Adobe Flash que se planeó para teléfonos móviles y otros dispositivos electrónicos como Chumby e iRiver.

- ***Ficheros***

Existen varios ficheros a la hora de trabajar con Flash. Se trata de ficheros que solamente se utilizan en tiempo de diseño para elaborar el resultado final.

FLA. Es el archivo que contiene los elementos multimedia: gráficos, animaciones, audios... que se utilizarán para crear el archivo final de salida. Es el archivo raíz de la película sobre el que luego podremos cargar otros elementos externos.

AS. Son archivos donde se recoge parte o la totalidad del código AS3 utilizado en la creación de una película Flash. Habitualmente contienen lo que en orientación a objetos se llaman clases.

Además de los archivos de trabajo se puede hacer uso en Flash de otros archivos para importar elementos externos dentro de la película. Por ejemplo es muy habitual el uso de archivos PNG y archivos JPG para utilizar bitmaps. Archivos de importación: **PNG, JPG, GIF, AI, FH, SWF, WAV, MP3,...**

Los ficheros que obtenemos como resultado después de la edición se llaman ficheros de salida. Existen diferentes tipos de ficheros de salida en función de lo que queramos hacer.

Los ficheros de salida más importantes de cara a programar con Flash son los siguientes:

SWF. Se trata de los ficheros que se utilizan para colocar contenido Flash en la web. Disponen de diversos métodos de compresión para los diferentes elementos que contiene (gráficos vectoriales, bitmaps, audio) lo que permite que el archivo sea más ligero.

Requiere de la instalación del plug-in de Flash Player en el navegador o la aplicación Flash Player para poder visualizarlo.

Si el usuario no dispone de Flash Player en ninguna de sus variantes se le puede facilitar un ejecutable independiente. En el sistema operativo Windows podemos crear un archivo **EXE**, para Mac se crea una aplicación (si creamos desde Windows un proyector para Mac la aplicación aparece comprimida en formato **HQX**). Hay que tener en cuenta que se genera una aplicación diferente para cada sistema operativo. Estos archivos incluyen dentro un reproductor de Flash, lo que hace que el archivo sea más pesado y difícil de manejar vía Internet. Por otro lado es uno de los archivos más utilizados a la hora de crear CDs o DVDs interactivos o ejecutables que se van a lanzar en local (juegos, instalaciones,...).

Además de estos ficheros podemos exportar desde Flash otro tipo de archivos como: vídeos **AVI**, animaciones **GIF**, secuencias de **JPGs**,...

- ***Problemas de Adobe Flash***

Complicaciones de la aceleración del video. Algunas veces Adobe Flash debe ser capaz de animar videos renderizados, necesitando un espacio de conversión de colores intermedio entre el video descodificado y el presentado, que un reproductor multimedia tradicional dejaría en hardware para hacer en una etapa posterior. Este paso intermedio la pipeline de aceleración se divide en dos. Dependiendo de las APIs de aceleración de hardware usado por el sistema operativo, hacer cualquiera de las partes por separado en el hardware puede ser incompatible o complicado por lo que es uno de los problemas que presenta Adobe Flash .

Adobe Flash está siendo bloqueado en algunos sitios web. Algunos sitios web dependían en gran medida de Adobe Flash, pero debido a la aparición de nuevas tecnologías y estándares como HTML5 flash está siendo cambiado por ellos y bloqueado en algunos de otros.

Oposición de Apple a su uso. La compañía Apple Inc., y mas su ex Presidente ejecutivo Steve Jobs, se negaron al uso de la tecnología Adobe Flash en cualquiera de sus dispositivos móviles iPod, iPhone e iPad. Esto se debió a varias razones, la principal es que es un software 100% propietario, es decir, cerrado a los aportes de terceros. También encontraron reticencias ante el antiguo historial de problemas de seguridad no resueltos por parte de Adobe. Y la última pero no menos importante es que al requerir decodificación por hardware, puede aumentar hasta en un 100% el uso de la batería de los dispositivos móviles.

Microsoft SilverLight

- ***Historia***

Silverlight es conjunto de tecnologías web que fue lanzada en septiembre de 2007 para competir con Macromedia Flash en la creación y animación de las aplicaciones web.

La versión 1.0 fue la de su nacimiento. Silverlight llega como una plataforma orientada a brindar una nueva experiencia de interacción entre usuarios y aplicaciones web desde el navegador. Esta primera versión trae consigo el primer intérprete de XAML, lenguaje descriptivo utilizado para la creación de interfaces, para un ambiente web, utilizando como lenguaje de programación a Javascript para lógica de la aplicación.

En la versión 2.0 Microsoft puso como nuevo corazón una implementación de la plataforma .NET, característica que pone a Silverlight en manos de desarrolladores experimentados en lenguajes como C#, Visual Basic e inclusive lenguajes como Python y Ruby (a través de IronPython e IronRuby), esto permite que la curva de aprendizaje para esta nueva tecnología se reduzca, y se puedan llevar aplicaciones existentes a Silverlight.

Con esta versión, se empiezan a incluir controles como el DataGrid, ListBox, Calendar, entre otros, disponibles para ser usados en nuestras aplicaciones, además de mejoras en la comunicación con protocolos a través de HTTP para facilitar el acceso a diferentes fuentes de datos mediante Internet, acercando esta versión, con nuevas capacidades para formularios y mejoras en el transporte de datos, al mundo de las aplicaciones LOB.

La versión 3.0 salió al mercado tras 2 años de maduración, y Microsoft aprovecha su capacidad de distribución de aplicaciones mediante internet para meterse de lleno en el mundo de aplicaciones LOB, con la capacidad de ejecutar las aplicaciones fuera del navegador (Out-Of-Browser), poniendo a disposición de los usuarios la aplicación directamente desde su escritorio, sin necesidad de abrir el navegador.

Incluye también nuevos controles (alrededor de 60 controles adicionales), desde nuevos controles “Layout”, controles como el AutoComplete, TreeView y DataGrid. Y, aunque no menos importante, empieza a aparecer como parte de Expression Studio 3, Sketch Flow

La versión 4.0 busca incrementar la eficacia de Silverlight en los ambientes corporativos.

Lo más importante de esta versión, son los WCF RIA Services, que permiten acceder de una manera más transparente a fuentes de datos, creando una capa de abstracción entre la aplicación Silverlight y el servicio web que expone los datos para alimentar la aplicación, esto combinado con las mejoras realizadas el sistema de DataBinding nos brinda nuevas posibilidades para aplicaciones tipo LOB. No solo el consumo de datos se vio mejorado, ya en esta versión, Silverlight puede leer y escribir datos para el usuario en las carpetas de "Mis Documentos", "Mi Música", "Mis Imágenes" y "Mis Videos".

Pero los datos en una aplicación tipo LOB no solo se almacena, también se muestran en forma de reportes, listos para ser impresos y presentados, todo de manera independiente del contenido de la pagina desplegada. Esta versión permite acceder a dispositivos y a características directamente al sistema operativo. Y no podían faltar las características incluidas en las herramientas de desarrollo, permitiendo desde Visual Studio 2010, editar completamente las interfaces de nuestras aplicaciones Silverlight, incluyendo el soporte para realizar DataBinding mediante Drag&Drop de componentes, selección de fuente de datos e integración con los recursos de estilos de Expression Blend.

En la última versión, la 5.0, se introdujeron importantes mejoras tanto en rendimiento (analizador XAML, latencia de operaciones sobe la red), como en funcionalidad que Microsoft introdujo en esta versión de Silverlight, también significa una mejora para la creación de aplicaciones tipo LOB.

El soporte multiventana llega para permitir que las aplicaciones Silverlight puedan segmentar la consulta de información, o inclusive aprovechar el uso de múltiples monitores para el despliegue de información. El poder ejecutar aplicaciones en "Full Trust" (usada, por ejemplo, para la lectura y escritura de archivos en cualquier parte dentro del sistema de archivos) dentro del navegado reduce aún más la línea que diferencia las aplicaciones Silverlight de las aplicaciones de escritorio, sin perder su capacidad de portabilidad en otros sistemas operativos.

Además de otras características que pueden hacerle la vida más fácil a los desarrolladores al crear este tipo de aplicaciones, como el permitir el proceso de Debug durante el Databinding, o realizar bindings en los Style Setters.

Adicionalmente, se incluyen controles para el despliegue y consulta de información de última generación como el PivotViewer.

(También podremos desarrollar aplicaciones en 3D con el uso del API de XNA al interior de esta última versión de Silverlight).

- ***Características***

Disponibilidad en Sistemas Operativos de escritorio. Microsoft Silverlight es un complemento de Microsoft que nos permite ejecutar aplicaciones enriquecidas en diferentes sistemas operativos. Windows y Mac OS X utilizan el complemento llamado Microsoft Silverlight. En cambio para Linux y FreeBSD Novell y Microsoft desarrollaron conjuntamente una aplicación de software libre llamada Moonlight que en 2012 fue abandonada debido a su poco uso por parte de los desarrolladores.

Disponibilidad en múltiples exploradores. Silverlight funciona sobre varias plataformas y múltiples exploradores especialmente Microsoft Internet Explorer. Silverlight supone una nueva forma de aprovechar los elementos multimedia en los principales navegadores tanto en MacOS como en Windows.

Desarrollo en ecosistema robusto y con desarrolladores experimentados Silverlight es actualmente desarrollado con una de las herramientas más importantes de Microsoft como es.NET lo que posibilita una nueva plataforma rica, segura y escalable para aprender de muchos desarrolladores experimentados.

Ofrece un modelo de programación flexible y coherente. Silverlight es compatible con lenguajes .NET como Visual Basic, C# y F#, y otras tecnologías web como AJAX, y lenguajes no desarrollados por Microsoft como Python, Ruby y que además se integra con las aplicaciones web existentes.

- ***Ficheros***

A diferencia de lo que pasaba con Adobe Flash, con Microsoft Silverlight no tiene unos ficheros intermedios de edición si no que la base de su programación es XAML (e**X**tensible **A**pplication **M**arkup **L**anguage, Lenguaje Extensible de Formato) y el acceso a los objetos está dado por C# y Visual Basic (aunque la versión 1.0 trabajaba a partir de JavaScript).

XAML es un lenguaje declarativo basado en XML, optimizado para describir gráficamente interfaces de usuarios visuales ricas desde el punto de vista gráfico el cual puede ser usado para marcar los gráficos vectoriales y las animaciones.

Estos gráficos vectoriales y capas serán compatibles con la integración de gráficos ampliables hasta cualquier tamaño, así como superposiciones con estilo de emisión televisiva para la adición de bandas y subtítulos (CC).

El formato de archivos multimedia unificado que soporta Microsoft Silverlight abarca desde contenidos HD hasta formatos para dispositivos móviles mediante Vídeo de Windows Media (WMV), la implementación de Microsoft del estándar SMPTE VC-1 de video, al igual que compatibilidad con audio WMA y MP3.

Microsft Silverlight contiene una herramienta de codificación eficaz para la publicación en vivo y a petición de experiencias de medios con Expression Media Encoder, que incluye la codificación con aceleración de hardware de WMV, VC-1, H.264, AAC y otros.

- ***Problemas de Microsoft Silverlight***

Problemas en navegadores no Microsoft. El mayor problema que tiene Microsoft Silverlight es su acoplamiento en navegadores como Chrome o Firefox. Debido a incompatibilidades que en más ocasiones de las deseables hace necesario desinstalar el complemento y volver a instalarlo.

Limitación del mercado. Debido al auge de las aplicaciones móviles el mercado de Microsoft Silverlight se ha visto muy reducido.

Problemas legales. De acuerdo con el organismo internacional sin fines de lucro, el Comité Internacional Europeo para la Interoperación entre Sistemas ("European Committee for Interoperable Systems") existe cierta preocupación de que con Silverlight Microsoft trata de introducir contenido en la red al que sólo se podría acceder desde la plataforma Windows. Argumentan que el uso de XAML en Silverlight se coloca para reemplazar al estándar HTML que es multi-plataforma. Efectivamente, si Silverlight llega a ser usado de forma amplia por mucha gente, existe el riesgo de que los usuarios tengan que llegar a comprar productos de Microsoft para poder acceder al contenido de internet.

Silverlight no usa estándares abiertos. Se ha criticado a Microsoft por no usar el estándar SVG (Scalable Vector Graphics) para Silverlight, lo cual, de acuerdo con Ryan Paul del sitio web 'Ars Technica', es consistente con la práctica de Microsoft de ignorar los estándares abiertos, como lo ha hecho en otros de sus productos.

JavaFX

- ***Historia***

Chris Oliver creó un proyecto llamado F3 (por Form Follows Function) el cual trabajaba en SeeBeyond Technology Corporation. En Septiembre de 2005 Sun Microsystems adquirió la compañía pasando Chris a ser su empleado.

En la conferencia JavaOne celebrada en San Francisco en 2007, Sun Microsystems presentó la plataforma JavaFX para ayudar a los desarrolladores de contenido y desarrolladores de aplicaciones a crear aplicaciones ricas en internet para dispositivos móviles, escritorios, televisores y otros dispositivos de consumo.

El 4 de diciembre del 2008 Sun Microsystems lanzó JavaFx 1.0, así como el plugin de JavaFx para NetBeans.

El 12 de febrero de 2009 se lanzó la versión JavaFX 1.1 que contenía como mejoras El apoyo oficial a JavaFX Mobile, mejoras de idioma y las mejoras de rendimiento y estabilidad con respecto a la versión anterior. El 2 de junio de de ese mismo año fue lanzada la segunda revisión de JavaFX.

En la versión JavaFX 1.2 se incluyó el soporte completo para Linux y Solaris, mejoras en la velocidad, permite la construcción de aplicaciones de escritorio, navegador y teléfono móvil.

En las versiones JavaFX 1.3 y JavaFX 1.3.1, fueron lanzadas 22 de abril y 21 de agosto de 2010 respectivamente, se incluyeron mejoras de rendimiento, se mejoró el soporte para controles de la interfaz de usuario o se introdujo una barra de progreso personalizada de inicio de la aplicación.

Hasta esa fecha Java FX no gozaba de mucha aceptación por parte de los desarrolladores. Fue a partir de la versión 2.0, el 10 de Octubre del 2011, cuando JavaFX empezó a tener más crédito por parte de los desarrolladores debido a un nuevo conjunto de API de Java, el cual proporcionó la apertura de las capacidades JavaFX a todos los desarrolladores de Java, sin necesidad de que aprender un nuevo lenguaje de scripting.

Además, en esta versión se incluyó también soporte para alto rendimiento, expresiones de enlace, encuadernación y varias mejoras en las librerías para el multithreading.

La versión JavaFX 2.1 fue lanzada el 27 de abril del 2012. E incluye la primera versión oficial para Mac OS X. que ha sido determinante en el desarrollo de aplicaciones multiplataforma. Otras mejoras fueron la adición de mejoras de interfaz de usuario, incluyendo mejoras en los controles de cuadro combinado, gráficos y las barras de menú y que el componente Webview ahora se puede activar para realizar llamadas a los métodos de Java.

La versión JavaFX 2.2 a parte de solucionarse problemas de la versión anterior se incluyeron nuevas características. Como el soporte para los estándares multimedia H.264 y AAC, soporte para Linux (incluyendo un complemento y webstart). Se añadieron Canvas y nuevos controles como Color Picker, Pagination. En esta versión se dio soporte al Streaming en vivo por HTTP y una API para la manipulación de fotografías e imágenes.

- ***Características***

Está basado en las mejores características de Java. La independencia de plataforma es una de las más conocidas de este. Una vez que los scripts de JavaFX han sido escritos estos pueden correr en cualquier plataforma, volviéndolo portable ya que Java se ha popularizado tanto. JavaFX pretende proveer seguridad a las aplicaciones.

Aplicaciones multiplataforma. Las aplicaciones de escritorio pueden ejecutarse en multitud de Sistemas Operativos como son Windows XP, Windows Vista, Windows 7, Mac OS, GNU/Linux y OpenSolaris.

Aprovecha de la mejor manera la plataforma de Java. Como lo es el completo acceso al API de Java, es ahí donde es posible usar las habilidades de programación de Java en la tecnología JavaFX. Ofrece también el mismo ambiente de desarrollo para los clientes web y no tiene ningún problema entre los exploradores.

Soporta por completo CSS de la misma manera que lo soporta HTML. Asimismo es posible integrar HTML y JavaScript en cualquier aplicación de JavaFX, fortaleciendo así la compatibilidad que tiene con su plataforma original: Java.

Swing y JavaFX pueden ser usados en la misma aplicación. Para que no sea necesario reescribir las aplicaciones creadas en Swing y que estas puedan ser extendidas de forma más fácil.

Provee una arquitectura unificada para el desarrollo de una aplicación. Una vez creada una aplicación es posible desplegarla en varios contextos, como una aplicación independiente, en un explorador web o por medio de Java Web Start. Y se piensan agregar más contextos en el futuro.

Licencia GPL v2. El compilador de JavaFx y el plugin para NetBeans se encuentran bajo licencia GPL v2 lo que hace que no se tengan importantes desembolsos a la hora de conseguir las licencias necesarias para empezar a desarrollar aplicaciones.

- ***Ficheros***

Al igual que Silverlight JavaFX no tiene ficheros intermedios como tiene Adobe Flash, si no que se basa en una arquitectura de aplicación diferente. Las aplicaciones de JavaFX se codifican como un programa java normal.

Tenemos nuestros ficheros de código fuente con extensión java que contienen las instrucciones que codifica la funcionalidad de la aplicación.

Para la parte visual de las aplicaciones de JavaFX se añadió la posibilidad de usar ficheros CSS especiales para este fin. Con ellos conseguimos darle un aspecto más atractivo a los diferentes elementos que hay en nuestra aplicación.

JavaFX soporta muchos de los formatos más utilizados (PNG, JPG, GIF, AI, FH, SWF, WAV, MP3,...) para mejorar la interactividad de las aplicaciones.

- ***Problemas de JavaFX***

Está basado en Java. De la misma manera en la que la dependencia de JavaFX en Java es una ventaja, también resulta una desventaja ya que se necesita tener todo el entorno de Java para que JavaFX funcione por completo. Otro aspecto relacionado es que no todo el contenido de JavaFX es libre, el runtime de JavaFX y el SDK seguirán estando bajo la Licencia de Código Binario de Java.

Inestabilidad en algunos Sistemas Operativos. Aún no existe una versión estable de JavaFX para los sistemas operativos Mac y Linux, lo que puede hacer que los desarrolladores busquen tecnologías más asentadas en dichos sistemas operativos.

Nuevas tecnologías que mejoran sus prestaciones. JavaFX es inferior a HTML5 ya que ofrece menos características que el estándar web. Aunque JavaFX es el futuro reemplazo de Swing, su desarrollo es diferente a las antiguas tecnologías de Java. Esto es consecuencia de agregar más características propias de las RIAs y no enfocarse a las aplicaciones de escritorio únicamente. Sin embargo esta no es una desventaja real ya que aun es más fácil para un desarrollador de Java aprender nuevos conceptos relacionados con algo que lleva usando desde hace mucho tiempo que aprender una nueva tecnología como es HTML5.

3.2. Tecnología elegida

Después de ver las ventajas y desventajas de cada una de las tres tecnologías que antes explicamos, llegamos a la conclusión que para lograr el objetivo principal del PFG (véase Objetivo) la tecnología que más se adapta a nuestras necesidades es JavaFX. Las razones son las siguientes:

La tecnología de JavaFX 2.2 nos ofrece la posibilidad de crear aplicaciones de escritorio tanto para Windows como para Mac OS X utilizando el lenguaje de programación Java. Al ofrecer la posibilidad de crear una aplicación tanto para Windows como para Mac OS X con el mismo código solucionamos el primer requisito del proyecto, que la aplicación sea multiplataforma..

El lenguaje Java con el que se codifica en JavaFX es muy parecido al que se usa para codificar en SWING o AWT, aunque con algunas nuevas particularidades y una estructura en los programas diferente a lo que sus predecesoras ofrecían, cuya curva de aprendizaje no es extremadamente difícil.

Una de las principales características de JavaFX está en la presentación y manipulación de elementos gráficos, lo cual nos da un plus a la hora de crear una interfaz de usuario vistoso y personalizable la cual haga posible que nuestro lector destaque sobre otros lectores del mercado. Para ello JavaFX se apoya en CSS en la parte visual, lo que separa la parte de presentación de la parte de negocio haciendo posible trabajar de manera separada en ambas partes con amplia independencia.

La agregación de efectos visuales es otra de las características más potentes que ofrece JavaFX ya que se consiguen una mejoría visual con respecto sus predecesoras SWING o AWT, llegando a un nivel parecido al que podría ofrecer AdobeFlash pero de una manera más orientada a la programación de aplicaciones de escritorio que al diseño publicista.

Uno de los requisitos que pretendíamos era la utilización del formato de fichero JSON para guardar la información de los libros. Debido este requisito la necesidad de que existieran varias librerías para el manejo de las estructuras JSON y Java proporciona de manera gratuita varias librerías de tratamiento de JSON. Esto hacía de JavaFX la mejor opción para el desarrollo de la aplicación.

3.3. Herramientas de desarrollo

Una vez decidido que JavaFX va a ser la tecnología en la que se va a basar nuestro desarrollo viene la decisión de que herramienta utilizar. Con JavaFx existen varias posibilidades de elección. Las dos herramientas de desarrollo más utilizadas son el IDE de NetBeans y el IDE de Eclipse.

El IDE NetBeans es una herramienta potente y cuya curva de aprendizaje es más sencilla y rápida que la de Eclipse. Tanto NetBeans como Eclipse tienen una comunidad de desarrolladores muy importante detrás de ellas, pero me decanté por NetBeans debido a que su integración con JavaFX estaba hecha por defecto como primera opción, evitando que se tardara mucho tiempo en su instalación, configuración antes de comenzar a desarrollarla aplicación en JavaFX.

El IDE Eclipse es una herramienta muy utilizada a nivel profesional, pero la unión entre Eclipse y JavaFX no es nativa y se necesita una configuración de JavaFX

con Eclipse con la que se necesita más tiempo en puesta en marcha que con NetBeans.

Gracias a que en diferentes asignaturas durante el grado se utilizó el IDE de NetBeans me sentía más cómodo al usarlo para desarrollar así que me decidí por esta herramienta en detrimento de Eclipse.

3.4. JSON

JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999.

JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un *objeto*, registro, estructura, diccionario, tabla hash, lista de claves o un array asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arrays, vectores, listas o secuencias.

Estas son estructuras universales; virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

En JSON, se presentan de estas formas:

Un *objeto* es un conjunto desordenado de pares nombre/valor. Un objeto comienza con { (llave de apertura) y termine con } (llave de cierre). Cada nombre es seguido por : (dos puntos) y los pares nombre/valor están separados por , (coma).

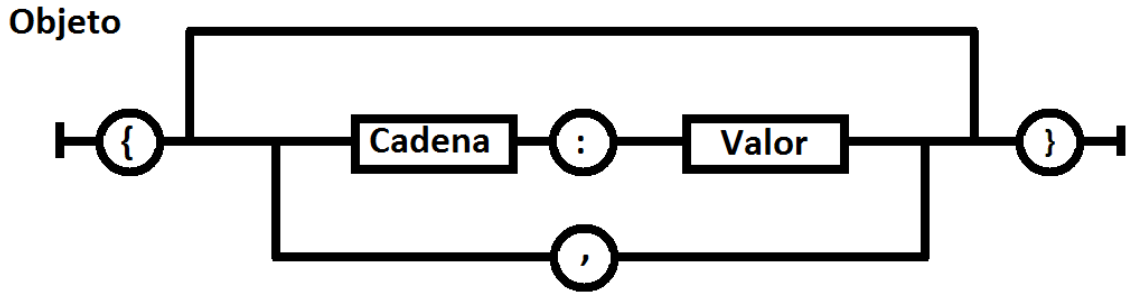


Ilustración 1Estructura Objeto JSON.

Un *array* es una colección de valores. Un arreglo comienza con [(corchete izquierdo) y termina con] (corchete derecho). Los valores se separan por , (coma).

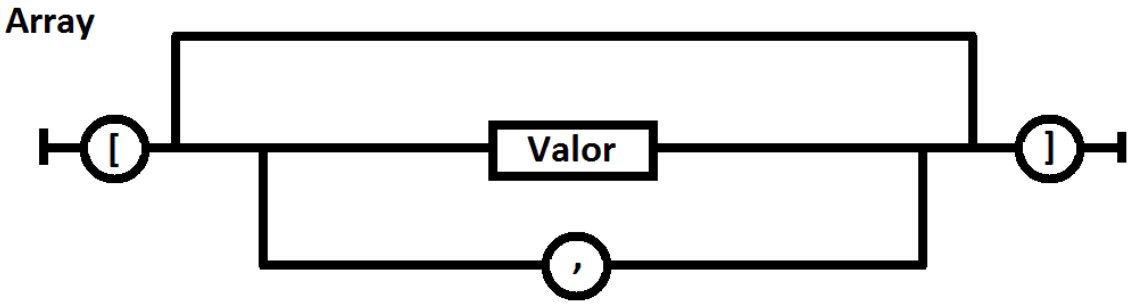


Ilustración 2 Estructura Array JSON.

Parte II: Desarrollo de la aplicación

4. Análisis de la aplicación

4.1 Introducción

En esta parte de la memoria vamos a pasar a detallar las fases que he seguido para desarrollar el lector. El ciclo de vida en cascada es en el que nos hemos basado: Análisis, Diseño, Implementación y Mantenimiento y Explotación. Como este lector aun no está en el mercado las dos últimas fases del ciclo de vida (Mantenimiento y Explotación) todavía no se han realizado y no aparecen en esta memoria.

4.2 Historias de usuario

En este apartado vamos a definir qué es lo que debe de hacer la aplicación del proyecto es decir; las funcionalidades de las que va a disponer el usuario a la hora de trabajar con la aplicación definiendo lo que podrán y lo que no podrán hacer con ella.

1. En la portada del lector	• El usuario puede:
	  • Seleccionar el libro que quiere leer.
	  • Cargar mas portadas de libros.
	• El usuario no puede:
	  • Cambiar el orden de los libros.

2. Navegación entre hojas	• El usuario puede: • Navegar entre las hojas de una en una.

	• El usuario no puede: • Cambiar ha hoja exacta poniendo el número.

3. Tratamiento capítulos	• El usuario puede:
	  • Ir hasta un capítulo del libro.
	• El usuario no puede:
	  • Crear nuevos capítulos.

4. Redimensionar hoja	• El usuario puede:
	  • Agrandar o empequeñecer la hoja actual.
	  • Reajustar el ancho de la hoja al ancho de la ventana.
	• El usuario no puede:
	  • Agrandar la imagen en un porcentaje exacto.

5. Tratamiento anotaciones de texto	• El usuario puede:
	  • Crear una anotación de texto en una hoja.
	  • Cargar y modificar anotación de texto ya creada.
	• El usuario no puede:
	  • Crear más de una anotación por hoja.
	  • Eliminar anotación de texto.

6. Tratamiento anotaciones gráficas	• El usuario puede:
	  • Cambiar el color de las anotaciones gráficas.
	  • Elegir anotación gráfica Rectángulo.
	  • Elegir anotación gráfica Subrayado.
	  • Elegir anotación gráfica Elipse.
	  • Dibujar anotación gráfica Rectángulo.
	  • Dibujar anotación gráfica Subrayado.
	  • Dibujar anotación gráfica Elipse.
	  • Eliminar anotación gráfica dibujada.
	  • Cambiar de posición anotación gráfica.
	• El usuario no puede:
	  • Deshacer la operación eliminar anotación gráfica.

7. Cambio de libro:	• El usuario puede: • Volver a la portada para cambiar de libro.

8. Funcionalidades de la ventana:	• El usuario puede:
	  • Guardar cambios introducidos.
	  • Minimizar la ventana.
	  • Maximizar o restaurar la ventana.
	• El usuario no puede:
	  • Mover la ventana por el escritorio.
	  • Cambiar el tamaño como queramos.

4.3 Casos de uso

4.3.1 Elección de libro

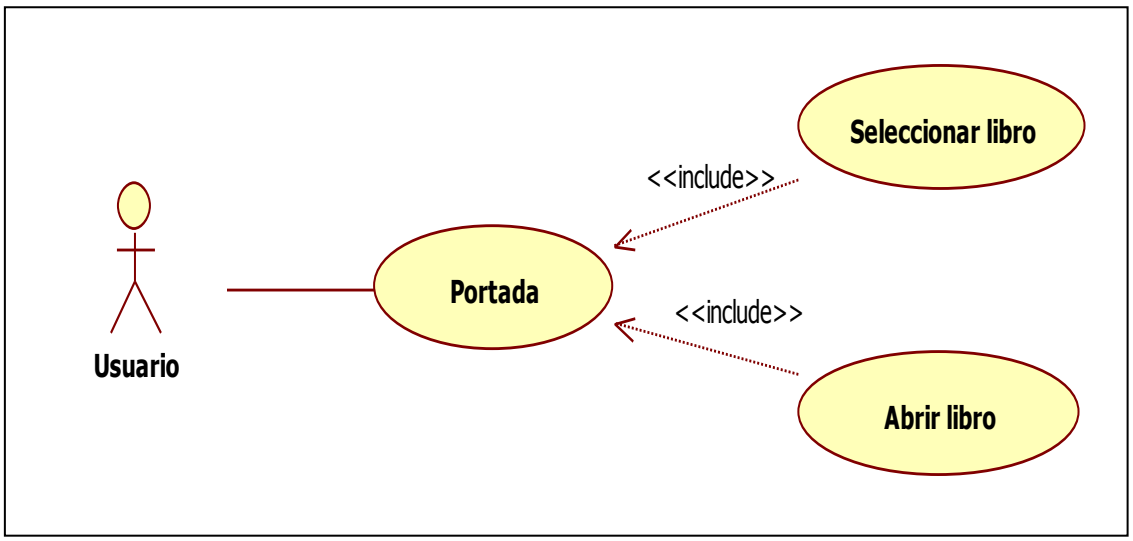


Ilustración 3Diagrama casos de uso Elección libro.

Actor

Usuario que desea utilizar la aplicación.

Casos de uso

- **Portada**: el usuario entra en la portada de la aplicación.
- **Seleccionar libro**: el usuario selecciona uno de los libros que aparecen en la portada del libro.
- **Abrir libro**: el usuario abre un libro seleccionado previamente

4.3.2 Añadir anotaciones de texto

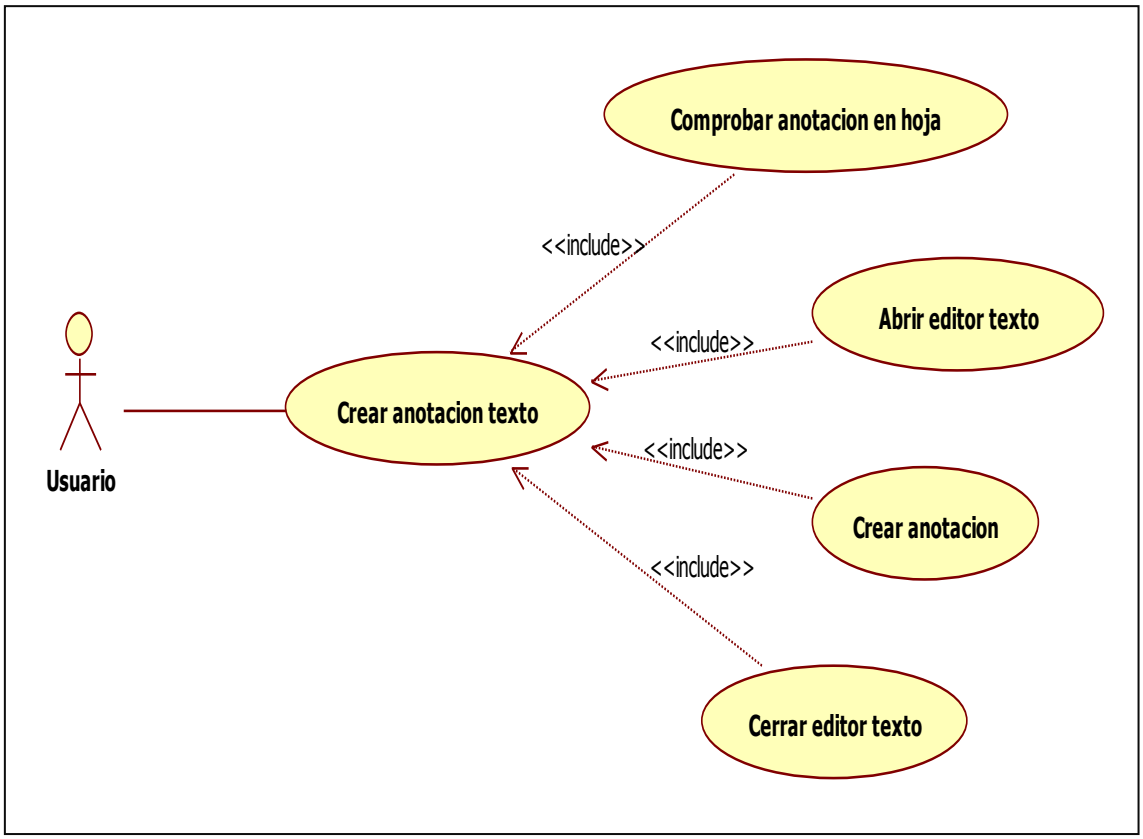


Ilustración 4Diagrama casos de uso Añadir anotación texto.

Actor

Usuario que desea utilizar la aplicación.

Casos de uso

- **Crear anotación texto**: el usuario desea crear una anotación de texto.
- **Comprobar anotación en hoja**: Se comprueba si ya existe una anotación de texto en esa hoja.
- **Abrir editor texto**: se abre un editor para que el usuario pueda escribir/modificar la anotación de texto.
- **Crear anotación**: se guarda la anotación de texto en el sistema.
- **Cerrar editor texto**: se cierra el editor de anotación de texto.

4.3.3 Dibujado de anotación gráfica

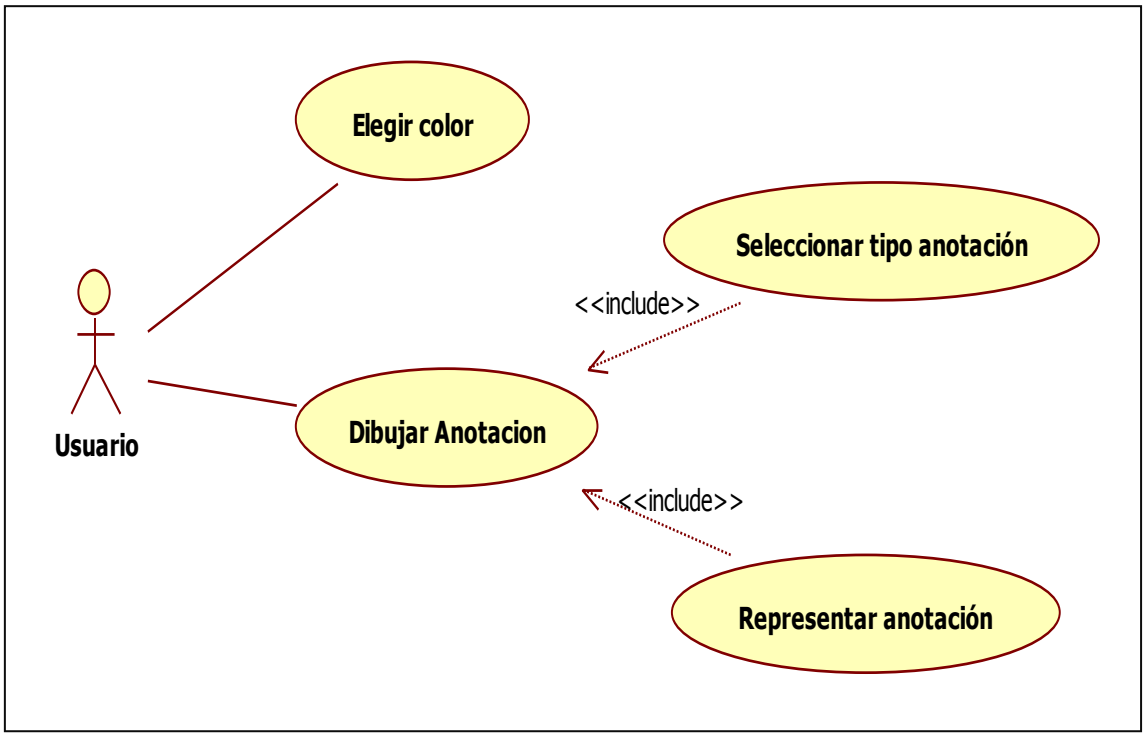


Ilustración 5Diagrama casos de uso Dibujar anotación gráfica.

Actor

Usuario que desea utilizar la aplicación.

Casos de uso

- **Elegir color**: el usuario elige el color de las siguientes anotaciones.
- **Dibujar anotación**: el usuario ejecuta la acción de dibujar una anotación gráfica.
- **Seleccionar tipo anotación**: el usuario elige que tipo de anotación desea plasmar en la hoja.
- **Representar anotación:** se dibuja sobre el canvas la anotación que el usuario a elegido en el lugar que él ha decidido.
- **Cerrar editor texto**: se cierra el editor de anotación de texto.

4.3.4 Eliminación anotación gráfica

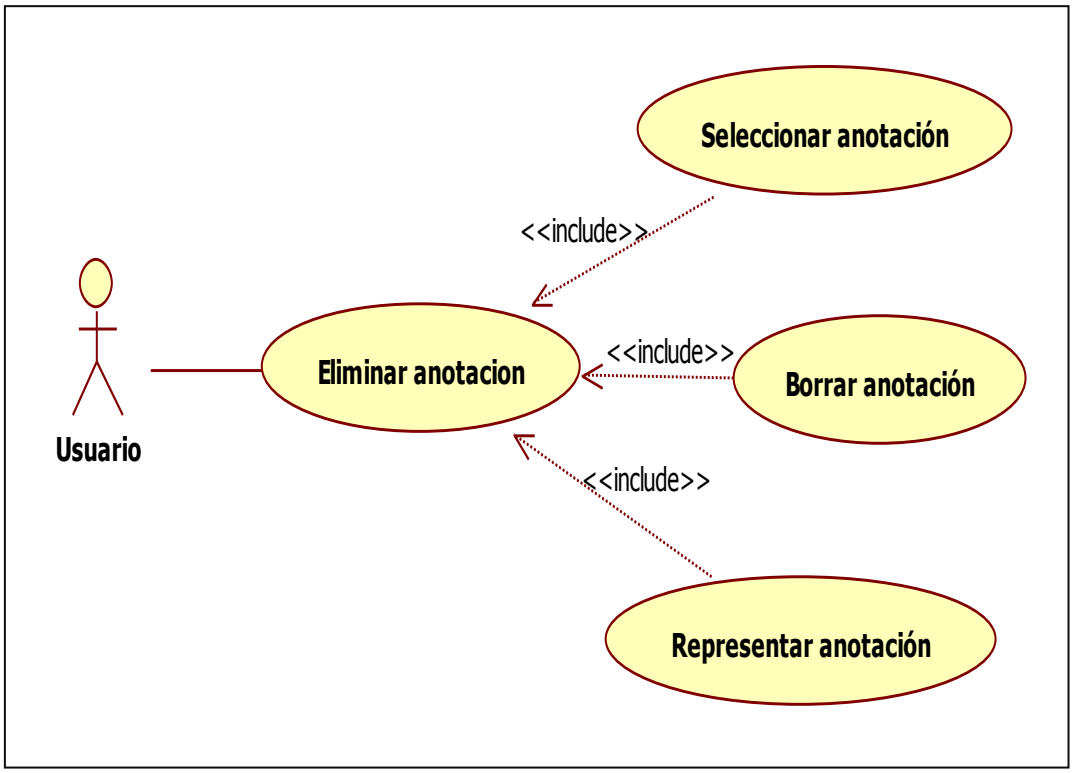


Ilustración 6Diagrama casos de uso Eliminar anotación gráfica.

Actor

Usuario que desea utilizar la aplicación.

Casos de uso

- **Eliminar anotacion**: el usuario puede eliminar una anotación hecha.
- **Seleccionar anotación**: el usuario elige que anotación desea eliminar de la hoja.
- **Borrar anotación:** se borra la anotación elegida por el usuario.
- **Representar anotación**: se muestra el canvas sin la anotación.

4.3.5 Mover anotación gráfica

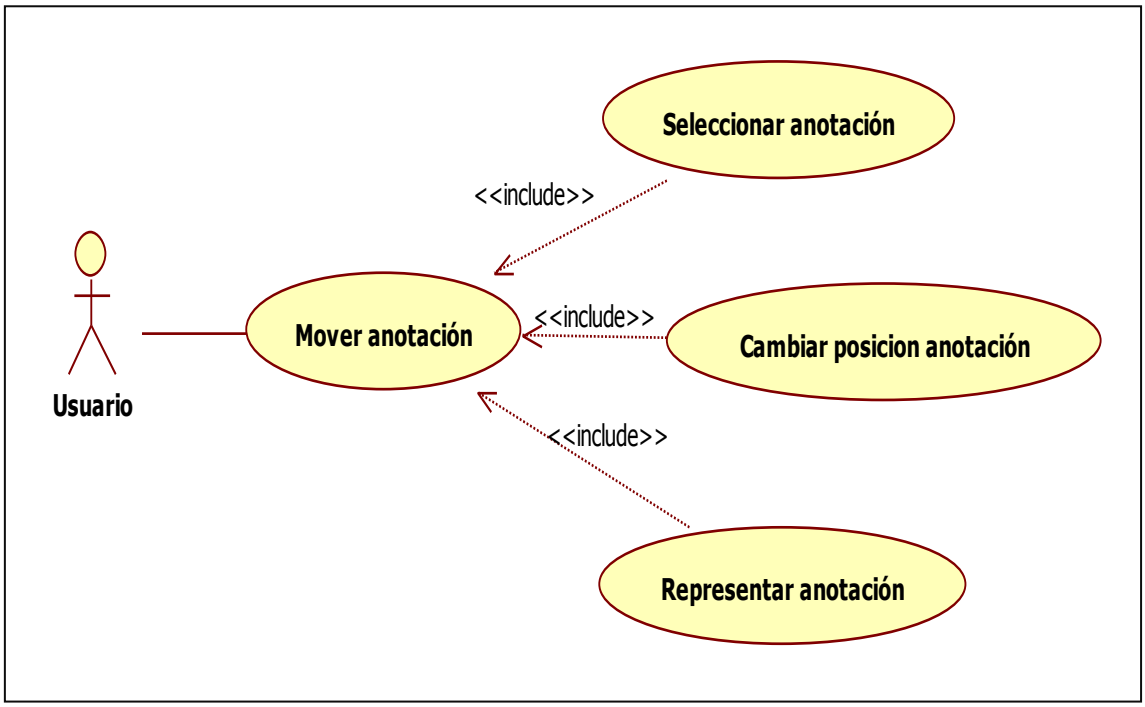


Ilustración 7Diagrama casos de uso Mover anotación gráfica.

Actor

Usuario que desea utilizar la aplicación.

Casos de uso

- **Mover anotacion**: el usuario puede eliminar una anotación hecha.
- **Seleccionar anotación**: el usuario elige que anotación desea mover en el canvas.
- **Cambiar posición anotación:** se cambia las coordenadas de la anotación elegida por el usuario.
- **Representar anotación**: se muestra el canvas con la anotación en su nuevo lugar.

5. Diseño de la aplicación

5.1 Introducción

Como bien explicamos antes estamos siguiendo el ciclo de vida en cascada y la siguiente fase en este ciclo de vida es la de diseño. En esta parte de la memoria es donde definiremos cual es la arquitectura de la aplicación, los interfaces y los componentes utilizados para mostrar como resultado el diseño final que llevamos a cabo para el PFG.

5.2 Definición de la estructura de una aplicación en JavaFX

Como ya comentamos en el apartado de estado del arte, JavaFX presenta diferencias considerables con las tecnologías predecesoras en programación de aplicaciones de escritorio de Java. Los cambios con respecto a SWING y AWT comienzan en el nuevo concepto de aplicación en el que se basa JavaFX.

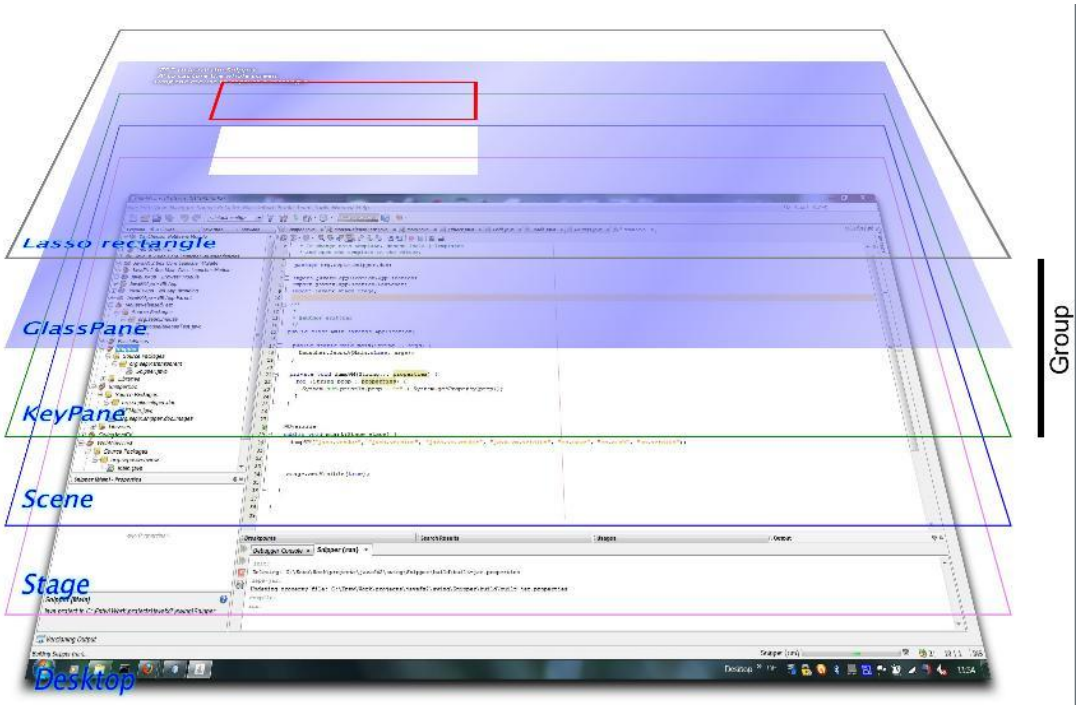


Ilustración 8 Estructura ventana JavaFX.

Como vemos en la imagen la base de la aplicación es ahora un Stage. Todo programa JavaFX que pretenda manejar material gráfico debe incluir un Stage en su nivel más alto, el Stage contendrá un objeto Scene, que a su vez contendrá una secuencia de Nodes. Por defecto el Stage tiene estilo StageStyle. DECORATED, que será visto de distinta forma según el sistema operativo en el que se ejecute la aplicación (Windows 7, Windows Vista, Windows XP, Mac OS). StageStyle, UNDECORATED eliminará la decoración de la ventana.

La siguiente capa que se ve es el Scene que es el root (la raíz) de todo el contenido en un escenario gráfico.

A partir del Scene se agregan los diferentes Nodes. La clase Node es la clase base para todos los objetos en el escenario gráfico. Se pueden añadir objetos Node (y subclases de Node) al escenario, especificar sus propiedades y aplicarles transformaciones. Esta clase tiene muchas propiedades que permiten personalizar su aspecto y comportamiento, como la visibilidad, los manejadores de eventos, opacidad, rotación, visibilidad y otras muchas propiedades. Además existen varias subclases de tipo Node (existen tres tipos: subclases predefinidas como Shape o ImageView con una funcionalidad predeterminada, la subclase Group aportará contenedores para más elementos tipo Node, y por último CustomNode nos permitirá personalizar nuestro propio tipo de Node según nuestras necesidades) para poder situar mejor la información en la interfaz. Para verlo desde otra perspectiva se muestra el siguiente diagrama.

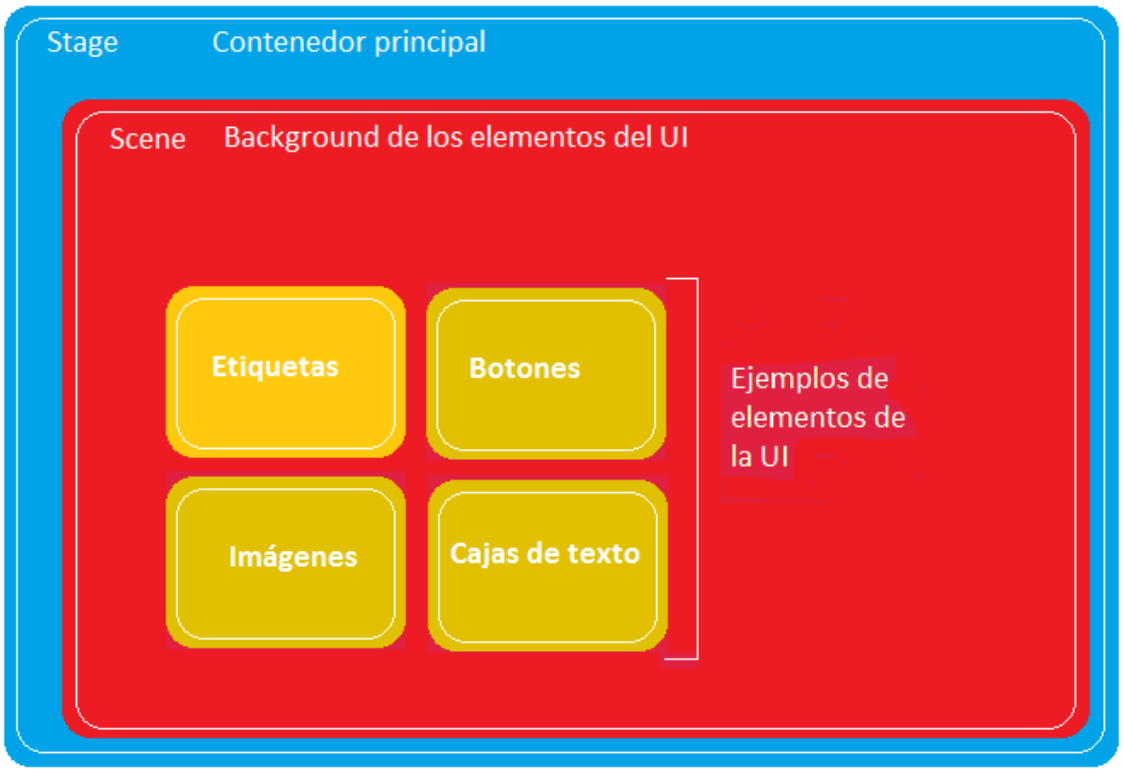


Ilustración 9 Estructura de los elementos en JavaFX.

En el ejemplo de la figura anterior, los elementos de la UI que se muestran (elementos de color naranja) pertenecen a esta clase.

Es curioso ver como algunos de los objetos que se manejan en este lenguaje se llaman Stage (escenario) o Scene (escena). Esto es así porque los creadores de la API lo modelaron de manera similar a una obra de teatro en la que los actores (Nodes) llevan a cabo la función delante de la audiencia. Con esta misma analogía, se entiende que aunque haya muchas escenas en las que trabajan los actores, todas ellas se realizan sobre un mismo escenario (relación de uno a muchos). Para Java Swing el símil de un Stage sería un JFrame o un JDialog. En estos contenedores había en su interior varios paneles en los cuales se insertaba a su vez los objetos que se quisieran mostrar en la interfaz. Estos paneles, en el caso de JavaFX corresponderían con las Scenes las cuales son capaces de albergar a varios Nodes (nodos) como se ha comentado previamente (ejemplos de nodos son: button, label, imageView, etc).

5.3 Patrones de diseño

Para la gente no especializada o con grandes conocimientos de programación los patrones de diseño son algo no muy usual así que definiremos que es un patrón de diseño a modo aclarativo.

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño resulta ser una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

En este PFG se han utilizado los patrones de diseño para solucionar los problemas que se han ido presentando durante todo el proceso de creación del software. No se han utilizado todos los patrones que existen, solo se han utilizado algunos que eran estrictamente necesarios para un buen funcionamiento del lector.

Una vez definidos pasaremos a enumerar y explicar los patrones de diseño utilizados en este PFG.

5.3.1 Patrón MVC

El patrón MVC (Modelo Vista Controlador) es usado para separar los datos y la lógica de negocio de la aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario

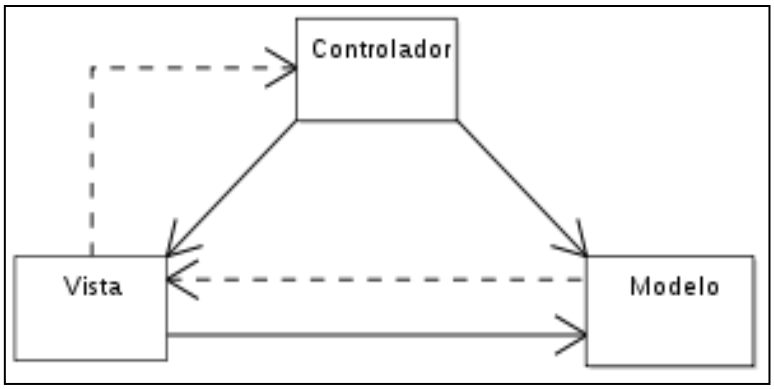


Ilustración 10 Estructura del MVC.

JavaFX nos provee de la posibilidad de construir nuestras aplicaciones usando el MVC con el uso una variante de XML propia de esta tecnología llamada FXML. El "Modelo" consiste en los objetos dentro del dominio de la aplicación, la "Vista" consiste en los ficheros FXML y el "Controlador" es el código Java que define el comportamiento de la interfaz gráfica de usuario (GUI) para interactuar con el usuario.

5.3.2 Patrón Command

Relacionado con el patrón MVC está el patrón Command, ya que este patrón se utilizó para encapsular las operaciones que se llevan a cabo en los controladores, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.

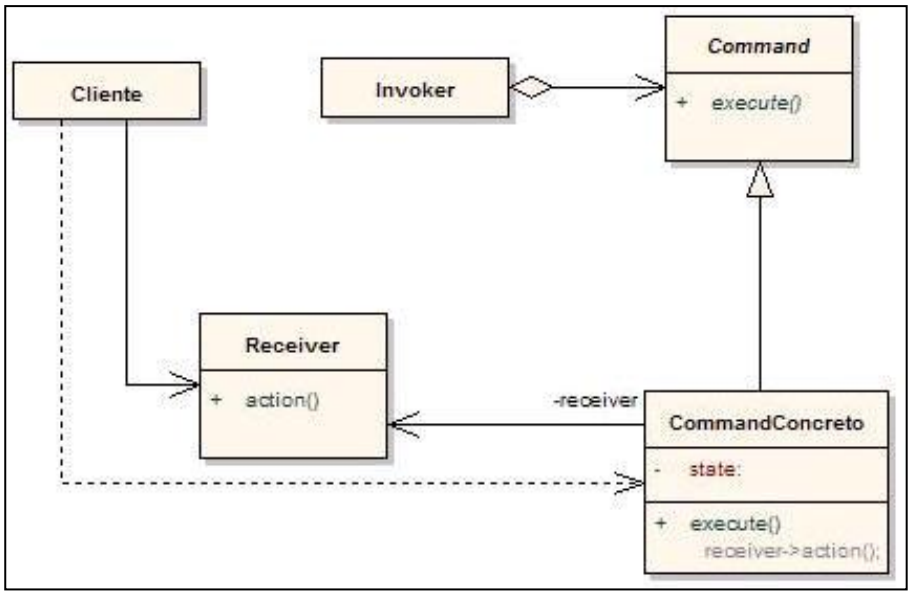


Ilustración 11 Estructura del patrón Command.

Con este patrón codificamos todas las funcionalidades de la interfaz del usuario para poder separar de manera más clara la Vista del Modelo.

También se ha utilizado este patrón para un uso interno dejando de forma más clara las diferentes funcionalidades que se pueden dar dentro de la aplicación.

5.3.3 Patrón Factory Method

Lo que se busca con este patrón es liberar al desarrollador sobre la forma correcta de crear objetos. Esto se consigue definiendo la interfaz de creación de un cierto tipo de objeto, permitiendo que las subclases decidan que clase concreta necesitan instancias.

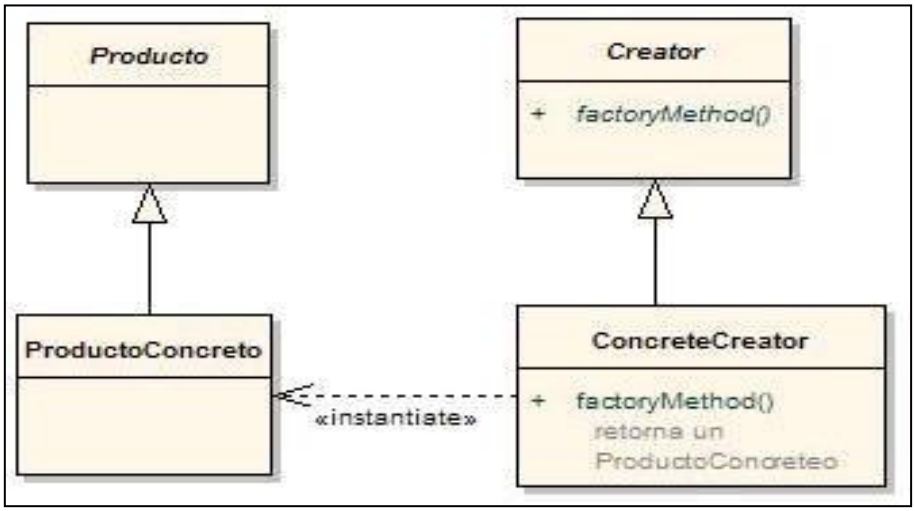


Ilustración 12 Estructura del patrón Factory Method.

El patrón FactoryMethod desarrolla una parte importante del PFG como es el tratamiento de la creación de anotaciones graficas dentro del canvas invisible. De esta forma logramos crear las anotaciones de una manera más sencilla y evitando problemas para el desarrollador al no tener que estar pendiente de cuál es la clase concreta necesita la instancia.

5.3.4 Patrón Prototype

El último patrón utilizado es el patrón Prototype. El patrón de diseño Prototype tiene como finalidad crear nuevos objetos duplicándolos, es decir, clonándolos una instancia creada previamente.

Este patrón especifica la clase de objetos a crear mediante la clonación de un prototipo que es una instancia ya creada. La clase de los objetos que servirán de prototipo deberá incluir en su interfaz la manera de solicitar una copia, que será desarrollada luego por las clases concretas de prototipos.

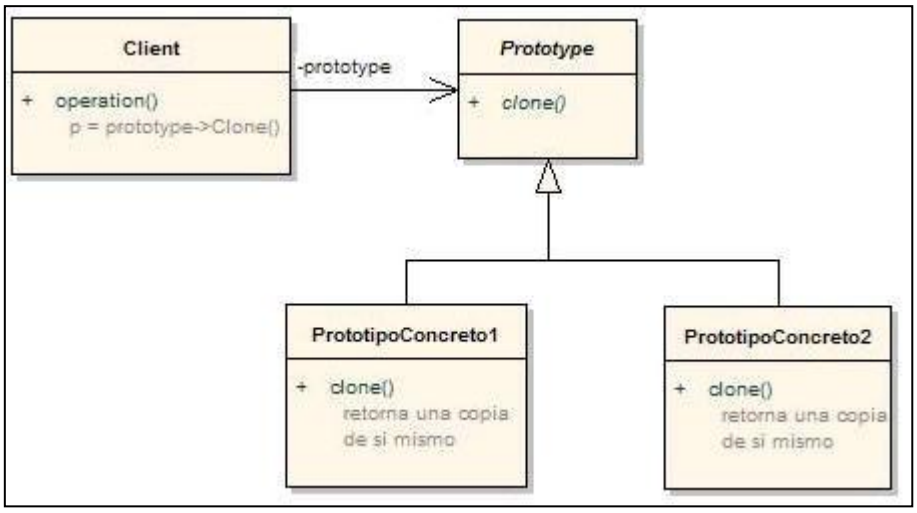


Ilustración 13 Estructura del patrón Prototype.

Este patrón se ha usado para la creación de las anotaciones dentro de la estructura de datos que guardará todos los datos de la biblioteca del usuario. Se eligió este patrón para conseguir más facilidad a la hora de crear los diferentes tipos de anotaciones gracias a la clonación de los prototipos.

5.4 Diseño de libros

5.4.1 Introducción

El lector de libros que construiremos tiene una serie de requisitos que tienen que ser solucionadas de una manera especial. El primer requisito es que la aplicación debe tener una separación entre las imágenes que conforman el libro y la información que se genera a raíz de las diferentes anotaciones que se pueden hacer en los libros.

Para ello los libros se dividirán es dos tipos de ficheros diferentes los ficheros de imágenes en los que cada libro se guardará de forma separada en un único fichero y el fichero de información, el cual guardará la información de todos los libros de un mismo usuario.

El segundo requisito es que la información de los diferentes libros tiene que guardarse en un formato particular, en este caso es JSON.

Detallaremos la estructura de cada uno de los tipos de ficheros y sus particularidades para que se vea la relación que existe entre ambos.

5.4.2 Fichero de imágenes del libro

Las imágenes del libro estarán contenidos en un único fichero binario en el cual se concatenarán una tras otra todas las imágenes de las que se conforma el libro final.

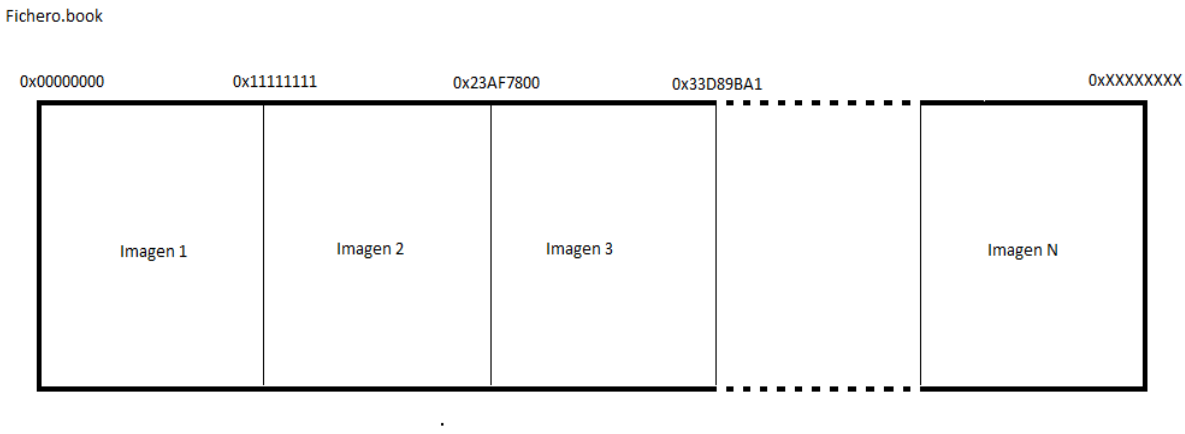


Ilustración 14 Estructura del fichero book.

En la figura vemos figura la forma que tendrá el fichero de imágenes de forma visual, y como se puede comprobar una parte muy importante de este fichero es el punto del fichero book donde termina una imagen y en cuál comienza la siguiente. Esta información nos permitirá decodificar la parte que deseemos y tener la foto para su tratamiento en el lector.

Esta información, a partir de ahora se llamará descriptor de página, se guardarán en el fichero JSON asociado al usuario para que el lector sepa en todo momento qué lugar ocupa cada foto en el fichero book del que se tiene que alberga las imágenes que componen las hojas del libro.

Fichero de información del libro

El fichero de información del libro guardará toda la información que sea necesaria para el funcionamiento del lector. Es la única entrada al sistema y la única salida del sistema. Todo lo que no se guarde en el JSON no se cargará en el lector.

Ahora explicaremos en qué consiste la estructura de ficheros JSON y detallaremos el formato del fichero de información de la aplicación para que el lector sepa para que sirve cada campo.

Formato JSON para nuestros libros

El fichero con nuestros libros será un array de objetos JSON. El formato será el siguiente:

```
{
    "title":"", Título del libro
    "author":" ", Autor del libro
    "year":" ", Año en el que fué escrito
    "numPages": , número de páginas total del libro
    "numBookmarks": , número de capítulos del libro
    "filename":" ", ruta donde se encuentra el fichero book con las imagenes,
    "pageDescriptors": Array con la información de los descriptores de página
    [
        {
            "init":, Bit de inicio de la imagen 1
            "end":  Bit de fin de la imagen 1
        },
        {
            "init": , Bit de inicio de la imagen 2
            "end": Bit de fin de la imagen 2
        }
    ],
    "bookmarks":
    [
        {
            "bookmarksTitle":" ", Título del capítulo
            "page": hoja que indica el inicio del capítulo
        },
        {
```

```
                "bookmarksTitle":" ", Título del capítulo
                "page": hoja que indica el inicio del capítulo
        }
],
"numNotes":, Numero de anotaciones



"notes": Array con las anotaciones
[
        {
        "type":"Rectangle",
        "pageNum":1,
        "initX":182.0,
        "initY":375.0,
        "width":231.0,
        "height":154.0,
        "scala":1.0,
        "color":
        {
                "red":0.501960813999176,
                "blue":0.501960813999176,
                "green":0.3019607961177826
        }
},
{
        "type":"Underline",
        "pageNum":1,
        "initX":95.0,
        "initY":157.0,
        "width":511.0,
        "height":45.0,
        "scala":1.0,
        "color":
        {
                "red":1.0,
                "blue":0.4000000059604645,
                "green":0.7019608020782471
        }
},
{
        "type":"Ellipse",
        "pageNum":1,
```

```
                "centerX":613.5,
                "centerY":121.0,
                "radiusX":67.5,
                "radiusY":38.0,
                "scala":1.0,
                "color":
                {
                        "red":0.0,
                        "blue":0.20000000298023224,
                        "green":0.20000000298023224
                }
        }
        ],
        "currentPage":, Página actual
        "lastColor": Último color con el que se hizo una anotación
        {
                "red":0.0,
                "blue":0.0,
                "green":0.0
        }
}
```

6. Implementación del sistema

6.1 Introducción

La última fase que vamos a analizar en esta memoria es la correspondiente a la fase de implementación en el ciclo de vida en cascada. La metodología española Métrica en su versión V3 define esta fase como el proceso en el que se genera el código de los componentes del Sistema de Información, se desarrollan todos los procedimientos de operación y seguridad y se elaboran todos los manuales de usuario final y de explotación con el objetivo de asegurar el correcto funcionamiento del Sistema para su posterior implantación.

En este apartado se va a explicar los patrones de diseño utilizados para solucionar diferentes problemas que se fueron planteando durante esta fase. También describiremos las estructuras internas más importantes de la aplicación y por último se describirá la interfaz gráfica de la herramienta explicando la función de cada uno de los elementos.

6.2 Patrones de diseño

En el apartado anterior os explicamos que son los patrones de diseño y que problemas solucionaban dentro del PFG. Ahora explicaremos como se han implementado en código los patrones que se incluyen en el PFG.

6.2.1 Patrón MVC

JavaFX nos provee de la posibilidad de construir nuestras aplicaciones usando el MVC con el uso una variante de XML propia de esta tecnología llamada FXML. El "Modelo" consiste en los objetos dentro del dominio de la aplicación, la "Vista" consiste en los ficheros FXML y el "Controlador" es el código Java que define el comportamiento de la interfaz gráfica de usuario (GUI) para interactuar con el usuario.

Basándonos en este patrón se ha desarrollado 4 vistas (2 principales y 2 ventanas auxiliares) con sus respectivos controladores. Cada una de las vistas son los elementos gráficos de los que se compone cada una de las vistas codificados con el lenguaje FXML.

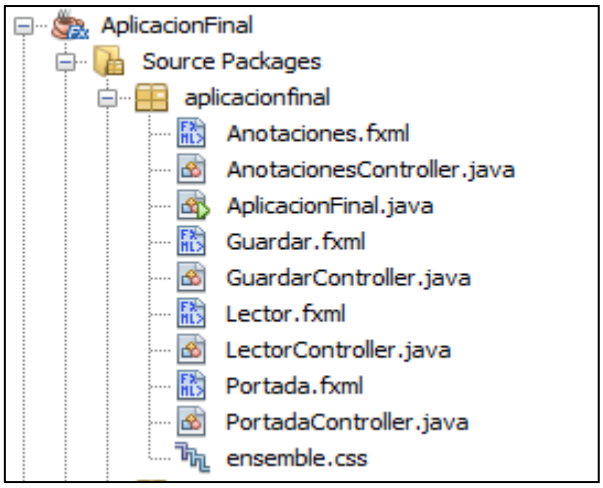


Ilustración 15 Elementos del MVC en JavaFX.

En Portada.fxml tenemos la vista de la portada que se le presenta al usuario nada más arrancar la aplicación. Portada.fxml contiene la estructura de los contenedores y los elementos que contiene la portada dejando el comportamiento de la GUI en el fichero PortadaController.java.

La vista lector son los elementos que el usuario ve cuando pasa a la pantalla de lectura y de edición de cualquier libro y está implementada en el fichero Lector.fxml y como anteriormente, la funcionalidad de los elementos de la GUI son codificados en el fichero LectorController.java.

Si echamos un vistazo al apartado 5-2, recordaremos que las aplicaciones en javafx se componen de un Stage y un Scene. Para estas dos vistas el Stage y el Scene son el mismo, lo único que se hace es cambiar la vista del scene cuando se quiere

cambiar de pantalla. Esta es una característica muy potente de JavaFX ya que permite tener cuantas vistas se quieran e ir cambiándolas en tiempo de ejecución según sea necesario.

Las vistas Anotaciones.fxml y Guardar.fxml son dos ventanas emergentes que aparecen en diferentes situaciones. Con Anotaciones.fxml lo que representamos son los elementos de la ventana que hace las veces de editor de anotaciones de texto. Con Guardar.fxml como bien dice su nombre será la ventana que se presentará cuando no se hayan guardado los cambios y se desee abandonar la aplicación.

6.2.2 Patrón Command

Dentro del PFG el patrón Command está codificado dentro del paquete patternCommand el cual contiene todos los comandos que pueden ser invocados desde los controladores y algunos comandos internos que se usan para clarificar el código.

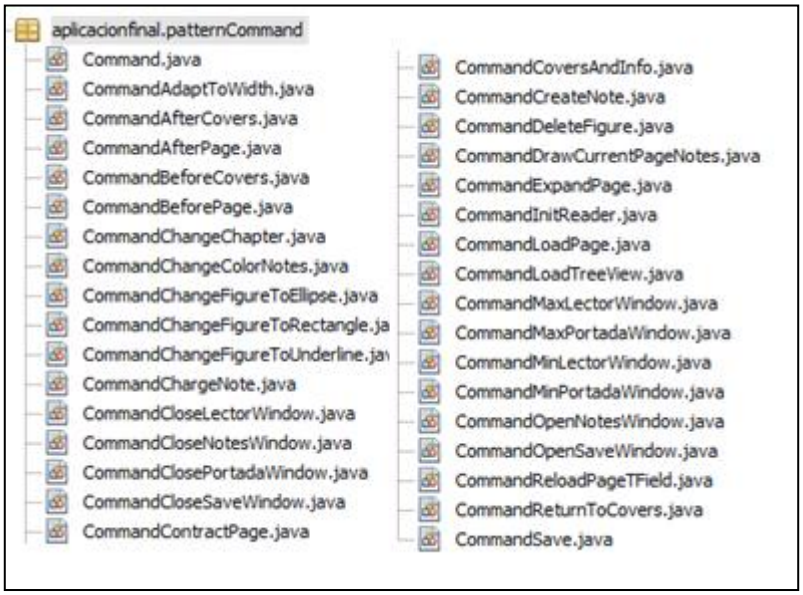


Ilustración 16 Elementos del patrón Command en JavaFX.

Como vemos en la figura existe un fichero llamado Command.java el cual contiene una interfaz que define los métodos que se implementarán en cada comando concreto. De esta forma definimos cuales son los métodos que deben compartir todos los comandos concretos.

El único método que contiene esta interfaz es el que tiene la funcionalidad que se ejecuta cuando se llama al comando concreto. En este fichero esta solo definido pero se debe implementar en cada uno de los comandos concretos.

El resto de ficheros que se ven en la imagen son todos los comandos concretos que se han desarrollado para este PFG los cuales desarrollan la funcionalidad que definimos en la interfaz del fichero Command.java.

Aunque una de las posibilidades que nos brinda este patrón es la funcionalidad de conseguir hacer y deshacer una operación, en este PFG se ha optado por dejar para versiones posteriores la implementación de esta funcionalidad del patrón Command.

6.2.3 Patrón Factory Method

Este patrón esta desarrollado en el paquete patternFactoryMethod y es utilizado para la funcionalidad de dibujado de las anotaciones gráficas.

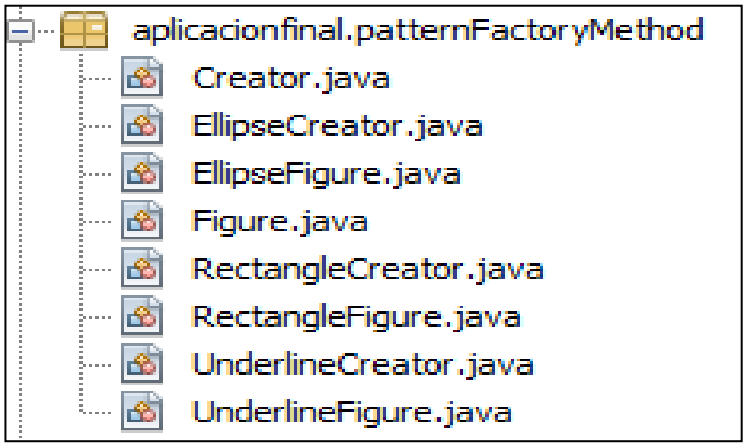


Ilustración 17 Elementos del patrón Factory Method en JavaFX.

Al implementar esta solución conseguimos que con un mismo Creator para todas las anotaciones podamos generar cualquiera de ellas sin necesidad de saber cuáles son todos sus detalles.

También se usa para poder aumentar dichas anotaciones según se vayan necesitando de una manera más sencilla y claramente definida para cualquier programador que se tenga que hacer cargo de la aplicación.

6.2.4 Patrón Prototype

Este patrón esta desarrollado en el paquete modelData y es utilizado para la creación de los objetos que contienen la información de las anotaciones realizadas en los libros.

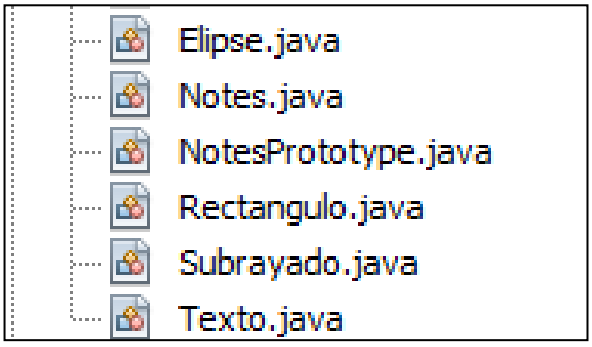


Ilustración 18 Elementos del patrón Prototype len JavaFX.

Como vemos en la imagen el fichero NotesPrototype.java es el fichero que implementa el patrón prototype. Para almacenar las diferentes notas utilizamos un arraylist de Notes con todos los tipos de que puede tener un libro. Dentro de ese arraylist guardamos una instancia de cada una de las diferentes anotaciones del libro (Elipse, Rectángulo, Subrayado y Texto).

Con el método Prototipo devolvemos un clon del tipo de anotación que queremos sin necesidad de conocer cuál es su interfaz.

Al implementar esta solución conseguimos que la generación de las anotaciones sea más sencilla dentro de la estructura BookCollection.

6.3 Estructuras de datos internas

En este apartado vamos a entrar en un poco mas de detalle para explicar algunas de las partes más importantes, y para que veáis como se han implementado algunas de las funcionalidades que se necesitaban para el correcto funcionamiento del PFG.

6.3.1 Estructura de clases de libros

La estructura interna que representa los datos de la colección de libros esta creada en base al fichero JSON que guarda toda la información de los libros y anotaciones que un usuario genera en el uso del lector. Esto es debido a que al iniciar la aplicación se lee del JSON del usuario toda la información guardada y se vuelca en el conjunto de clases que dan formato a toda esa información.

La clase BookCollection representa la información de todos los libros que el usuario tiene en lectura. El objeto que se inicializa al principio de la ejecución es pasado mediante inyección de dependencias por toda la aplicación y se modifica la información según se va necesitando. Únicamente se vuelca la información de nuevo en el fichero JSON cuando el usuario usa la opción de guardar dentro de la UI del lector. BookCollection está compuesta por un conjunto de libros los cuales están representados en la clase Book.

Book representa todos los elementos de un libro, entre los que están sus descriptores de página (DescriptorPage), sus capítulos (Chapter) y sus anotaciones (Notes) de los cuales hablaremos luego.

Cada objeto de Book contiene toda la información necesaria para el correcto funcionamiento del lector. Los tipos de datos más importantes que hay en la clase Book son:

DescriptorPage: Esta clase representa el byte de inicio y byte de fin de cada una de las hojas de las que se compone el libro dentro del fichero book asociado. Si esta información no fuera correcta no se podría mostrar la imagen correcta en el lector.

Chapter: Esta clase representa la información necesaria para cada uno de los capítulos en lao que se divide el libro. La información que se guarda es el nombre del capítulo y el número de hoja donde empieza dicho capitulo. Con esta información podemos ir al capítulo que queramos en el lector.

Notes: Esta clase representa el global de las anotaciones que podemos realizar dentro de cualquiera de los libros. Estas son de diferente tipo (texto, rectángulo, elipse, subrayado) y se pueden añadir más tipos según las necesidades que se vayan detectando. Cada anotación tiene una serie de atributos comunes y otros que son particulares.

6.3.2 Estructura ElementosUI

La clase ElementosUI contiene una copia de todos los elementos gráficos que hay en las ventanas del lector. Estas copias se pasan por toda la aplicación haciendo los cambios necesarios en este objeto y luego pasándolo a la parte del controlador donde se vuelve a representar con los cambios ya realizados.

Esta funcionalidad se utiliza para hacer más independientes la parte de vista y la de modelo. Al no poder acceder la una a la otra en cada una de las operaciones que se hacen habitualmente.

6.4 Partes de código

En este apartado se van a mostrar y comentar algunas de las partes de código más interesantes y relevantes en el desarrollo de la aplicación.

6.4.1 Lectura del fichero JSON

Este es el fragmento de código que se utiliza para leer del fichero JSON y guardar la información en la estructura interna que será guardada en el objeto del tipo BookCollection. Para la extracción de datos se ha optado por el uso de la librería json-simple-1.1.1.jar que es de uso libre y bastante utilizada en proyectos que necesitan el tratamiento de ficheros JSON.

```
public ArrayList<Book> read() {
    Book libro;
    String jsonFile = "";
    Object obj;
    JSONArray jsonArrayDescriptores;
    JSONObject jsonDescriptores;
    JSONArray jsonArray;
    JSONObject jsonLibros;
    JSONArray jsonArrayCapitulos;
    JSONObject jsonCapitulos;
    JSONArray jsonArrayAnotaciones;
    JSONObject jsonAnotacion;
```

```
JSONObject jsonColor;
JSONObject jsonLastColor;
JSONParser parser = new JSONParser();
FileReader fr = null;
String linea;

try {
   fr = new FileReader(ruta);
   BufferedReader entrada = new BufferedReader(fr);
   while ((linea=entrada.readLine())!=null)
   {
      jsonFile = jsonFile.concat(linea + " ");
   }
} catch (FileNotFoundException e) {
   System.out.println(e.getMessage());
} catch (IOException e) {
   System.out.println(e.getMessage());
} finally {
   try {
      if (fr != null) {
         fr.close();
      }
   } catch (IOException e) {
      System.out.println(e.getMessage());
   }
}

if (!jsonFile.equals("")) {
   try {
      obj = parser.parse(jsonFile);
      jsonArray = (JSONArray) obj;
      for (int cont = 0; cont < jsonArray.size(); cont++) {
         libro = new Book();
         jsonLibros = (JSONObject) jsonArray.get(cont);
         libro.setTitulo(jsonLibros.get("title").toString());
         libro.setAutor(jsonLibros.get("author").toString());
         libro.setAnyo(jsonLibros.get("year").toString());
         libro.setPaginas(
               Integer.parseInt(jsonLibros.get("numPages").toString()));
         libro.setCapitulos(
               Integer.parseInt(jsonLibros.get("numBookmarks").toString()));
         libro.setFichero(jsonLibros.get("filename").toString());
         jsonArrayDescriptores =
```

```
        (JSONArray) jsonLibros.get("pageDescriptors");
    libro.inicializaDescriptores(libro.getPaginas());
    for (int i = 0; i < libro.getPaginas(); i++) {
      jsonDescriptores = (JSONObject) jsonArrayDescriptores.get(i);
      libro.setDescriptorInicio(
              Integer.parseInt(jsonDescriptores.get("init").toString()), i);
      libro.setDescriptorFin(
              Integer.parseInt(jsonDescriptores.get("end").toString()), i);
    }
    jsonArrayCapitulos = (JSONArray) jsonLibros.get("bookmarks");
    libro.inicializaCapitulos(libro.getCapitulos());
    for (int i = 0; i < libro.getCapitulos(); i++) {
      jsonCapitulos = (JSONObject) jsonArrayCapitulos.get(i);
      libro.setCapituloTitulo(
              jsonCapitulos.get("bookmarksTitle").toString(), i);
      libro.setCapituloPage(
              Integer.parseInt(jsonCapitulos.get("page").toString()), i);
    }
    int numAnotaciones = Integer.parseInt(
                          jsonLibros.get("numNotes").toString());
    if (numAnotaciones != 0) {
      jsonArrayAnotaciones = (JSONArray) jsonLibros.get("notes");
      for (int i = 0; i < numAnotaciones; i++) {
        jsonAnotacion = (JSONObject) jsonArrayAnotaciones.get(i);
        switch (jsonAnotacion.get("type").toString()) {
          case "Text":
            libro.setAnotacionTexto(
                jsonAnotacion.get("type").toString(),
                Integer.parseInt(jsonAnotacion.get("pageNum").toString()),
                jsonAnotacion.get("text").toString());
            break;
          case "Underline":
            jsonColor = (JSONObject) jsonAnotacion.get("color");
            libro.setAnotacionSub(
                jsonAnotacion.get("type").toString(),
                Integer.parseInt(jsonAnotacion.get("pageNum").toString()),
                Double.parseDouble(jsonAnotacion.get("initX").toString()),
                Double.parseDouble(jsonAnotacion.get("initY").toString()),
                Double.parseDouble(
                      jsonAnotacion.get("width").toString()),

Double.parseDouble(jsonAnotacion.get("height").toString()),
                Double.parseDouble(jsonAnotacion.get("scala").toString()),
```

```
                    Double.parseDouble(jsonColor.get("red").toString()),
                    Double.parseDouble(jsonColor.get("blue").toString()),
                    Double.parseDouble(jsonColor.get("green").toString()));
                break;
            case "Rectangle":
                jsonColor = (JSONObject) jsonAnotacion.get("color");
                libro.setAnotacionRect(
                    jsonAnotacion.get("type").toString(),
                    Integer.parseInt(jsonAnotacion.get("pageNum").toString()),
                    Double.parseDouble(jsonAnotacion.get("initX").toString()),
                    Double.parseDouble(jsonAnotacion.get("initY").toString()),
                    Double.parseDouble
                        (jsonAnotacion.get("width").toString()),
                    Double.parseDouble(
                        jsonAnotacion.get("height").toString()),
                    Double.parseDouble(jsonAnotacion.get("scala").toString()),
                    Double.parseDouble(jsonColor.get("red").toString()),
                    Double.parseDouble(jsonColor.get("blue").toString()),
                    Double.parseDouble(jsonColor.get("green").toString()));
                break;
            case "Ellipse":
                jsonColor = (JSONObject) jsonAnotacion.get("color");
                libro.setAnotacionElip(
                    jsonAnotacion.get("type").toString(),
                    Integer.parseInt(jsonAnotacion.get("pageNum").toString()),
                    Double.parseDouble(
                        jsonAnotacion.get("centerX").toString()),
                    Double.parseDouble(
                        jsonAnotacion.get("centerY").toString()),
                    Double.parseDouble(
                        jsonAnotacion.get("radiusX").toString()),
                    Double.parseDouble(
                        jsonAnotacion.get("radiusY").toString()),
                    Double.parseDouble(jsonAnotacion.get("scala").toString()),
                    Double.parseDouble(jsonColor.get("red").toString()),
                    Double.parseDouble(jsonColor.get("blue").toString()),
                    Double.parseDouble(jsonColor.get("green").toString()));
                break;
        }
      }
    }
    libro.setPaginaEnLectura(
        Integer.parseInt(jsonLibros.get("currentPage").toString()));
```

```
            jsonLastColor = (JSONObject) jsonLibros.get("lastColor");
            Color lastColor = new Color(
                  Double.parseDouble(jsonLastColor.get("red").toString()),
                  Double.parseDouble(jsonLastColor.get("green").toString()),
                  Double.parseDouble(jsonLastColor.get("blue").toString()),
                  1.0);
            libro.setLastColor(lastColor);
            libros.add(libro);
          }
        } catch (ParseException ex) {
          Logger.getLogger(JSonBooks.class.getName()).log(Level.SEVERE, null, ex);
        }

      } else {
        //ERROR al leer fichero
      }
      return libros;
  }
```

6.4.2 Escritura del fichero JSON

Aquí tenemos la segunda función más importante en el tratamiento de los ficheros JSON, la función de escribir en el fichero JSON. En este fragmento como se puede comprobar se utilizan las funciones de la librería json-simple mediante las cuales creamos el fichero con los datos guardados en el objeto BookColection.

```
public void write(ArrayList<Book> libros) {
    JSONArray arrayColeccionLibros = new JSONArray();
    for (int cont = 0; cont < libros.size(); cont++) {
      JSONArray array_descriptores_pagina = new JSONArray();
      JSONArray array_capitulos = new JSONArray();
      JSONArray array_anotaciones = new JSONArray();

      Map libro = new LinkedHashMap();
      libro.put("title", libros.get(cont).getTitulo());
      libro.put("author", libros.get(cont).getAutor());
      libro.put("year", libros.get(cont).getAnyo());
      libro.put("numPages", libros.get(cont).getPaginas());
      libro.put("numBookmarks", libros.get(cont).getCapitulos());
      libro.put("filename", libros.get(cont).getFichero());

      for (int i = 0; i < libros.get(cont).getPaginas(); i++) {
```

```
            Map descriptores_pagina = new LinkedHashMap();
            descriptores_pagina.put("init", libros.get(cont).getDescriptorInicio(i));
            descriptores_pagina.put("end", libros.get(cont).getDescriptorFin(i));
            array_descriptores_pagina.add(descriptores_pagina);
        }

        for (int j = 0; j < libros.get(cont).getCapitulos(); j++) {
            Map capitulos = new LinkedHashMap();
            capitulos.put("bookmarksTitle", libros.get(cont).getCapituloTitulo(j));
            capitulos.put("page", libros.get(cont).getCapituloPage(j));
            array_capitulos.add(capitulos);
        }

        for (int k = 0; k < libros.get(cont).getSizeAnotaciones(); k++) {
            Map descriptores_anotacion = new LinkedHashMap();

            Map color = new LinkedHashMap();

            descriptores_anotacion.put("type",
libros.get(cont).getAnotacion(k).getTipo_anotacion());
            descriptores_anotacion.put("pageNum",
libros.get(cont).getAnotacion(k).getPagina());

            switch (libros.get(cont).getAnotacion(k).getTipo_anotacion()) {
                case "Text":
                    Texto text = (Texto) libros.get(cont).getAnotacion(k);
                    descriptores_anotacion.put("text", text.getText());
                    array_anotaciones.add(descriptores_anotacion);
                    break;
                case "Underline":
                    Subrayado sub = (Subrayado) libros.get(cont).getAnotacion(k);

                    color.put("red",   sub.getRed());
                    color.put("blue",  sub.getBlue());
                    color.put("green", sub.getGreen());

                    descriptores_anotacion.put("initX",  sub.getInitX());
                    descriptores_anotacion.put("initY",  sub.getInitY());
                    descriptores_anotacion.put("width",  sub.getWidth());
                    descriptores_anotacion.put("height", sub.getHeight());
                    descriptores_anotacion.put("scala",  sub.getScala());
                    descriptores_anotacion.put("color",  color);
                    array_anotaciones.add(descriptores_anotacion);
```

```
            break;
        case "Rectangle":
            Rectangulo rect = (Rectangulo) libros.get(cont).getAnotacion(k);

            color.put("red",   rect.getRed());
            color.put("blue",  rect.getBlue());
            color.put("green", rect.getGreen());

            descriptores_anotacion.put("initX",  rect.getInitX());
            descriptores_anotacion.put("initY",  rect.getInitY());
            descriptores_anotacion.put("width",  rect.getWidth());
            descriptores_anotacion.put("height", rect.getHeight());
            descriptores_anotacion.put("scala",  rect.getScala());
            descriptores_anotacion.put("color",  color);
            array_anotaciones.add(descriptores_anotacion);
            break;
        case "Ellipse":
            Elipse elip = (Elipse) libros.get(cont).getAnotacion(k);

            color.put("red",   elip.getRed());
            color.put("blue",  elip.getBlue());
            color.put("green", elip.getGreen());

            descriptores_anotacion.put("centerX",  elip.getCenterX());
            descriptores_anotacion.put("centerY",  elip.getCenterY());
            descriptores_anotacion.put("radiusX",  elip.getRadiusX());
            descriptores_anotacion.put("radiusY", elip.getRadiusY());
            descriptores_anotacion.put("scala",  elip.getScala());
            descriptores_anotacion.put("color",  color);

            array_anotaciones.add(descriptores_anotacion);
            break;
        default:
            break;
    }
}

libro.put("pageDescriptors", array_descriptores_pagina);
libro.put("bookmarks", array_capitulos);
libro.put("numNotes", libros.get(cont).getSizeAnotaciones());
libro.put("notes", array_anotaciones);
libro.put("currentPage", libros.get(cont).getPaginaEnLectura());
```

```
        Map lastColor = new LinkedHashMap();
        lastColor.put("red",   libros.get(cont).getLastColor().getRed());
        lastColor.put("blue",  libros.get(cont).getLastColor().getBlue());
        lastColor.put("green", libros.get(cont).getLastColor().getGreen());
        libro.put("lastColor",lastColor);

        arrayColeccionLibros.add(libro);
    }


    String jsonText = JSONValue.toJSONString(arrayColeccionLibros);
    System.out.print(jsonText);

    //Escribimos el fichero json
    FileWriter fichero = null;
    PrintWriter pw;
    try {
        fichero = new FileWriter(ruta);
        pw = new PrintWriter(fichero);

        pw.print(jsonText);
    } catch (IOException e) {
    } finally {
        try {
            // Nuevamente aprovechamos el finally para
            // asegurarnos que se cierra el fichero.
            if (null != fichero) {
               fichero.close();
            }
        } catch (IOException e2) {
        }
    }
}
```

6.4.3 Decodificación de hoja

En este fragmento vemos como se extrae una imagen del fichero .book que contiene todas las imágenes del libro. Es una de las partes más importantes puesto que extraemos en un buffer todos los bytes que componen la imagen basándonos

en el inicio y el fin que nos delimitan los descriptores de páginas guardados en el JSON.

Una vez conseguido extraer ese buffer se devuelve como un ByteArrayInputStream el cual se pasa a un objeto Image y se consigue la imagen con todas sus funcionalidades de manera transparente para el usuario.

```
private ByteArrayInputStream getPageImage(String path, int inicio, int fin, int size)
{
    int leido = 0;
    FileInputStream fis;
    BufferedInputStream bi;
    this.size = size;// fin - inicio;
    this.bufferInicial = new byte[fin];
    this.bufferFinal = new byte[this.size];

    try {
      // Se abre el fichero
      fis = new FileInputStream(path);
      bi = new BufferedInputStream(fis);

      /* Leo desde el principio hasta la posicion final que necesito */
      leido = bi.read(bufferInicial, 0, fin);

      /* Quito la parte sobrante del buffer inicial leido antes y me quedo
       * con la imagen que necesito */
      for (int j = inicio; j < fin; j++) {
        bufferFinal[j - inicio] = bufferInicial[j];
      }

      fis.close();
      bi.close();

    } catch (Exception e) {
      e.printStackTrace();
    }

    return new ByteArrayInputStream(this.bufferFinal, 0, size);
  }
```

6.5 Manual de usuario

En este apartado trataremos de mostrar el funcionamiento del lector que se ha desarrollado en este PFG. De una manera visual y su explicación textual iremos desgranando cada una de las posibles acciones que están implementadas en este proyecto.

Para empezar vamos a ver la pantalla principal que el usuario puede ver cuando se ejecuta, esta es la portada de todos los libros que el usuario tiene cargados.

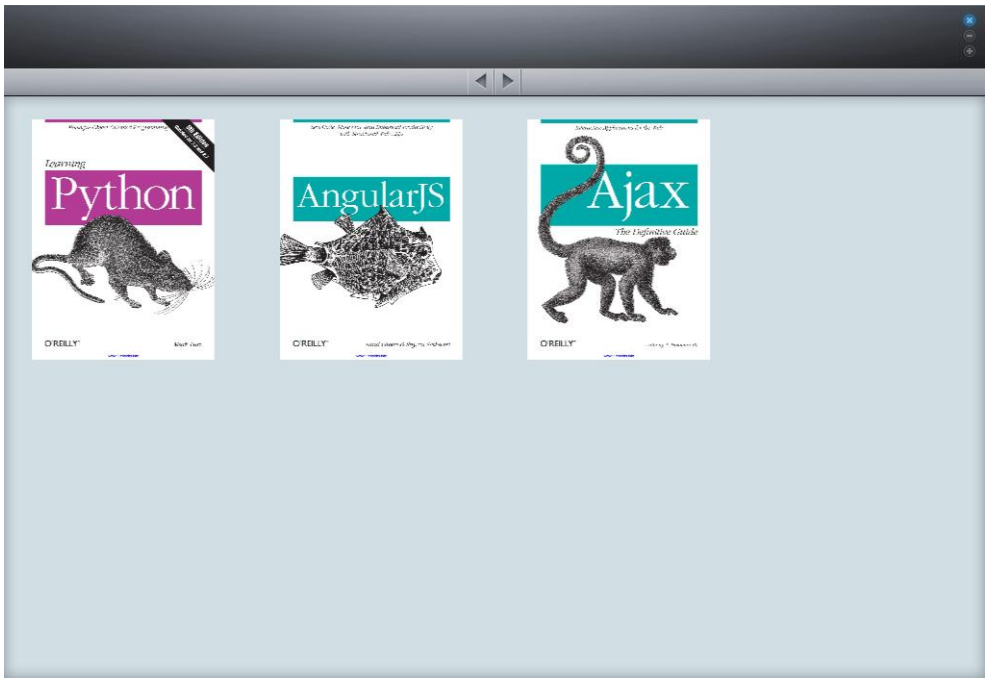


Ilustración 19 Portada del lector

Como se puede ver el diseño es muy simple sin grandes extravagancias, estilo Mac OS. Se aprecian fácilmente las partes en las que se divide la ventana. En la parte superior una barra de ventana algo más gruesa de lo habitual donde tenemos los botones típicos de cualquier ventana, botón de cerrar, minimizar y restaurar/maximizar. En segundo término vemos una barra de herramientas en la cuál ahora mismo solo hay dos botones los cuales nos permitirán navegar entre todas las portadas de los libros que posee el usuario. Por último se ve el área más grande que es donde se muestran todas las portadas de los libros.

Una vez el usuario posa el ratón encima de cualquiera de las imágenes se mostrará la información del libro (Titulo, Autor, Número de páginas, Año de edición) y un botón en el cual si pulsamos se abrirá el libro.

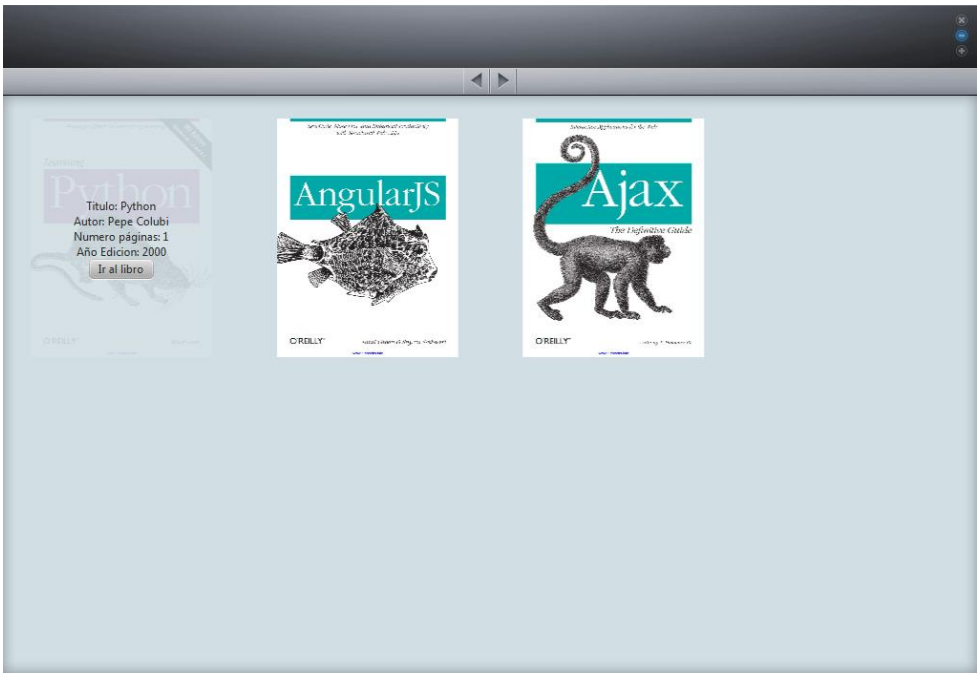


Ilustración 20 Portada del lector con animación.

Cuando hemos pulsado el botón "Ir al libro" pasamos a la ventana de lectura, en la cual en primera instancia la misma distribución de la ventana que en la anterior imagen.

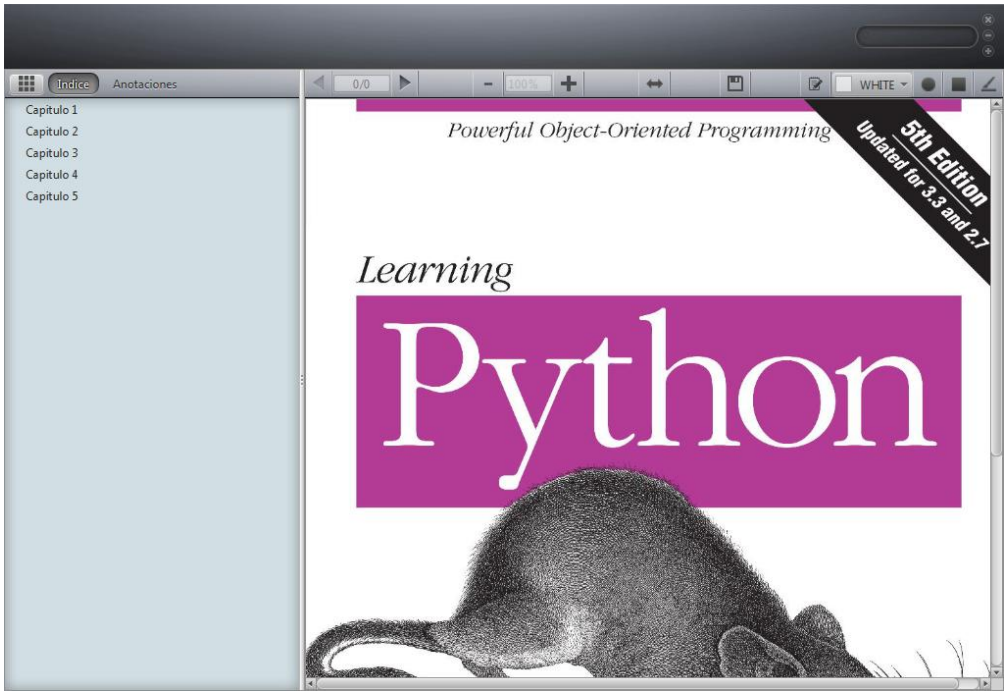
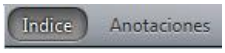


Ilustración 21 Pantalla lector y editor de libros.

Como se ve en la imagen, existen dos zonas nuevas y multitud de nuevos botones los cuales nos permitirán realizar todas las acciones necesarias para el tratamiento de las anotaciones, la navegación por el libro, etc.

A continuación explicaremos de izquierda a derecha cuales son las funcionalidades de los botones que hay en la barra de herramientas.

El primer botón  tiene como funcionalidad volver a la portada. Una vez que hayamos terminado con la edición de este libro o simplemente queramos cambiar a otro, pulsaremos este botón y pasaremos a la ventana de las portadas de los libros.

Los dos botones 

son un ToggleButton mediante el cual controlamos que es lo que aparece en el área que hay debajo de estos botones. Cuando está presionado el botón "Índice" se muestran los capítulos en los que está dividido el libro. En cambio si el que esta pulsado es el botón "Anotaciones" se nos muestran las anotaciones que hay hechas en cada hoja.

Los siguientes botones son 

los cuales se usan para navegar entre las hojas del libro. Como se puede deducir cada uno indica hacia dónde va a pasar la hoja. Tal y como está planteado el proyecto ahora mismo no se puede cambiar el valor de la hoja actual y que se transite hasta la misma.

Para realizar el redimensionamiento de las hojas del libro que se muestran en el lector usamos los botones 

los cuales agrandan o reducen la hoja en el porcentaje que sale en el TextBox. Al igual que antes no está implementado el cambio del porcentaje de manera manual.

También para ayudarnos con el redimensionamiento el botón  nos permite ajustar el ancho de la hoja al del área donde se representa la hoja.

El siguiente botón  como bien se puede ver es con el cual guardamos los cambios que hemos realizado en el libro.

Ahora comienza el área de los botones para la creación de anotaciones. El primero que encontramos  es con el cual abrimos la ventana de creación de anotaciones de texto. El siguiente 

es para seleccionar el color que queremos utilizar en las anotaciones gráficas. El botón  es la anotación elipse, el siguiente  es para la anotación rectángulo y el último  la anotación subrayado.

Ahora vamos a explicar la funcionalidad de las anotaciones gráficas. Para dibujar tenemos que pulsar en cualquiera de los botones de anotaciones gráficas (elipse, rectángulo o subrayado) y después pinchamos con el ratón encima de la hoja y arrastramos para dibujar. Mientras recorremos la hoja comprobamos que se va dibujando y una vez soltamos se queda la anotación dibujada con el color que habíamos elegido. Cuando ya se ha dibujado la anotación si se pincha encima de

ella el contorno se resalta. Y si pinchas y arrastras se mueve la anotación por toda la hoja hasta que se suelta y tiene una nueva posición. La hoja lleva de anotaciones puede quedar así.

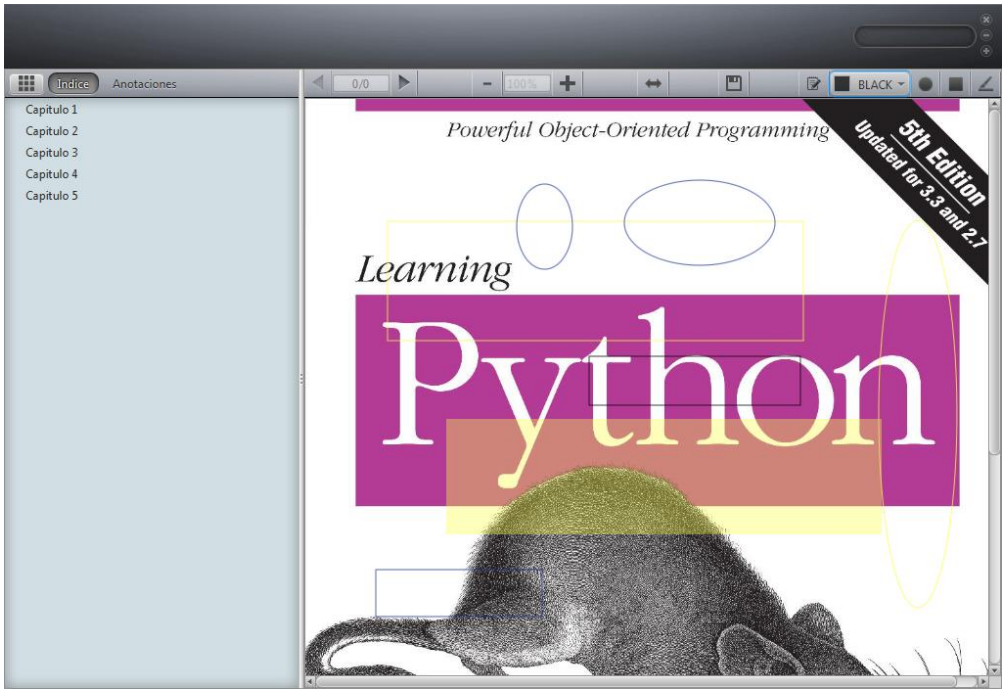


Ilustración 22 Libro con anotaciones gráficas.

Para generar anotaciones de texto solo tenemos que pulsar en el botón de generar anotaciones y nos saldrá una nueva ventana con la cual podamos crear nuestra anotación con formato HTML.

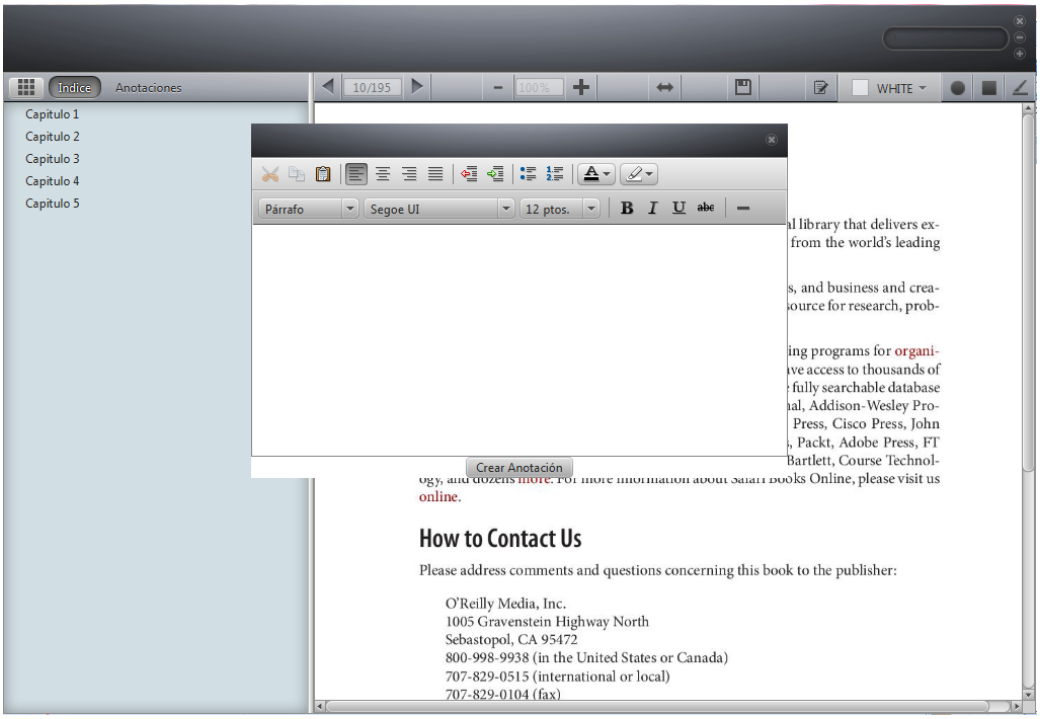


Ilustración 23 Editando anotación de texto.

Una vez hayamos escrito todo lo que deseamos y de darle el formato que queramos pulsamos el botón “Crear Anotación” con el cual se cerrará esta ventana y se generará el correspondiente elemento en el área izquierdo de la ventana principal y aparece una señal de que hay una anotación en la hoja.

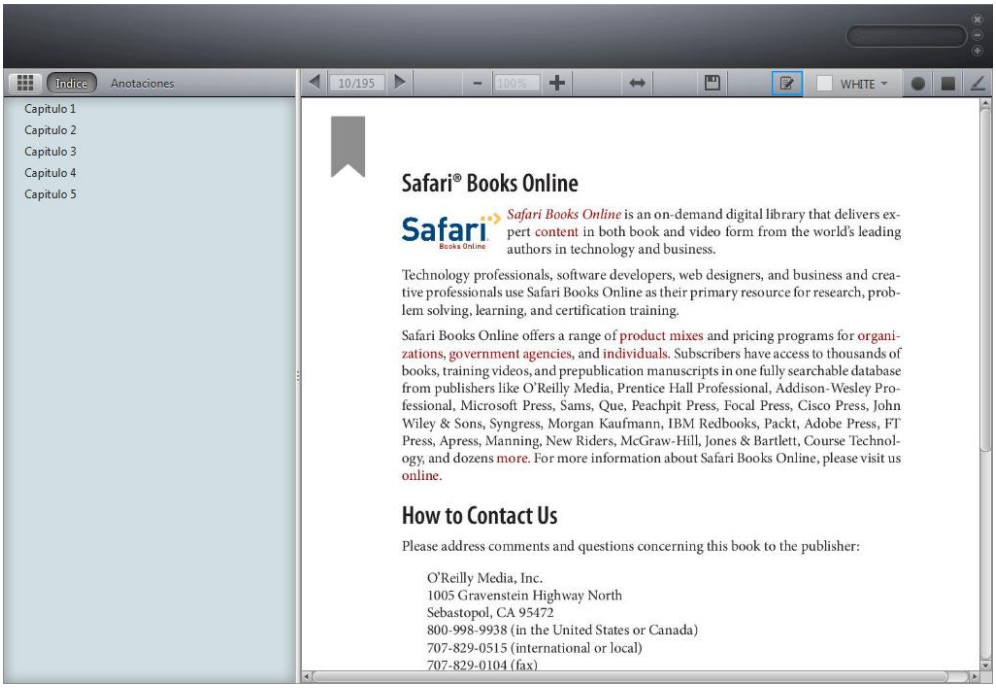


Ilustración 24 Nota de texto recién creada.

Si volvemos a pulsar el botón en una hoja donde anteriormente ya se había creado una anotación nos saldrá la ventana de creación de notas de texto pero con el texto que ya teníamos escrito.

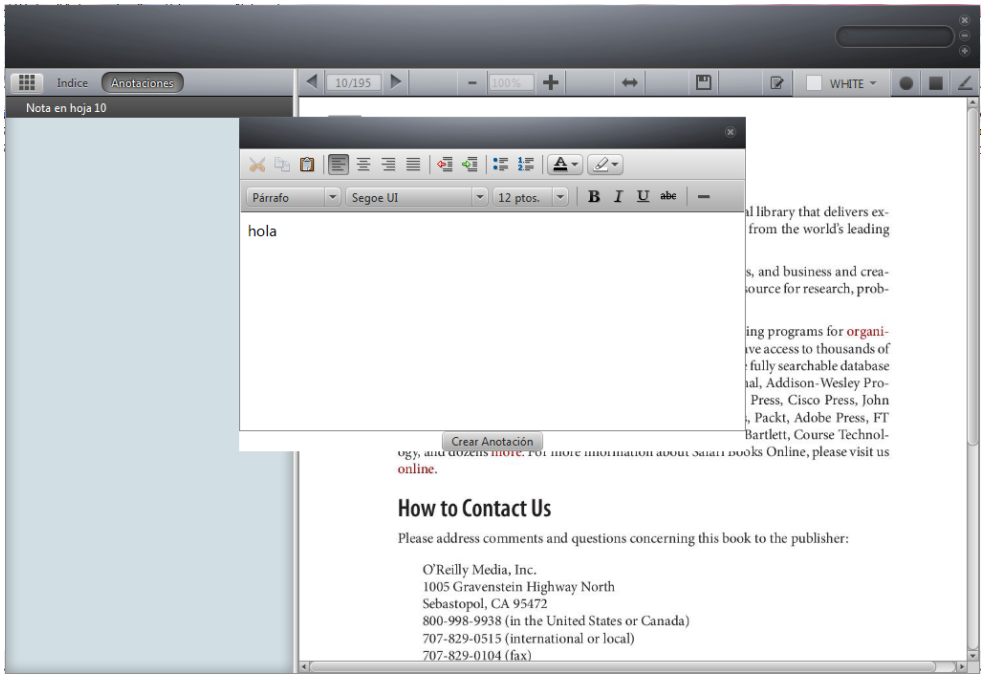


Ilustración 25 Reeditar hoja de texto.

Parte III: Cierre de proyecto

7. Presupuesto

Datos sobre el presupuesto que ha sido necesario a lo largo de este proyecto. Se han calculado los costes de trabajo personal, gastos en material y gastos indirectos. Para el supuesto se ha usado el horario laboral de la empresa para la que trabajo y los salarios medios de las ofertas de trabajo que se ha buscado por internet.

7.1 Coste personal

En este apartado, se detalla el proceso operacional seguido para la obtención del salario por hora de trabajo de los diferentes roles participantes en el proyecto. Los resultados obtenidos se presentan, posteriormente, en forma de tabla.

- **Días laborables / semana**: 5 días.
- **Horas trabajadas / día**: 8.42 horas de lunes a jueves – 6.30 horas viernes.
- **Horas trabajadas / semana**: 40 horas.
- **Salario neto por persona al mes** (el mismo para todos los roles): 1.300 €.
- **IRPF** (17% sobre el salario neto): 221 €.
- **Seguridad Social** (40% sobre salario neto + IRPF): 741 €.
- **Salario bruto por persona / mes** (Salario neto + IRPF + Seguridad Social): 2.041 €.
- **Meses laborables**: 12 meses.
- **Salario bruto por persona y año** (suponemos 14 pagas): 28.574 €.
- **Horas en un año** = 1.742 horas.
- **Coste bruto por persona y hora** = Salario bruto por persona y año / horas en un año = 28.574€ / 1.742 horas = 16,40 €/h

7.2 Coste por hora de los roles

Se llama roles a cada uno de los papeles que a lo largo del proyecto se desempeñan. Se ha puesto el mismo coste a todos, aunque esto no es cierto en proyectos reales.

Roles	Coste por hora
Jefe de proyecto (JP)	16,40
Analista (ANA)	16,40
Diseñador (DIS)	16,40
Programador (PRO)	16,40
Responsable de pruebas (RP)	16,40

Tomando una metodología de trabajo típica en cascada de análisis, diseño, implementación y pruebas, la siguiente tabla, recoge el conjunto de horas trabajadas en cada rol en el proyecto. Se supone una jornada laboral de 8 horas.

Actividad	JP	ANA	DIS	PRO	RP	Total
Análisis	60 h	60 h				**120 h**
Diseño		60 h	60h			**120 h**
Implementación			90 h	90 h		**180 h**
Pruebas				60 h	60 h	**120 h**

Una vez expuesto los datos de las dos tablas anteriores, se detalla a continuación, el coste por rol y por actividad desempeñados dentro del proyecto.

Actividad	JP	ANA	DIS	PRO	RP	Total
Análisis	984 €	984 €				**1968€**
Diseño		984 €	984 €			**1968€**
Implementación			1.476€	1.476€		**2952€**
Pruebas				984 €	984 €	**1968€**

7.3 Coste del material utilizado

Material	Coste
Equipo informático	500 €
Licencias software	0 €
TOTAL	**500 €**

7.4 Costes indirectos

Este gasto es obtenido como el 15% de los costes directos, esto es el 15% sobre coste personal + coste del material.

7.5 Resumen de costes del proyecto

Se calcula ahora el coste final del proyecto, partiendo de los datos obtenidos en los apartados anteriores.

Conceptos	Costes
Coste personal	8856€
Coste material	500
Coste indirecto	1403,40 €
TOTAL	**10759,40 €**

8. Conclusiones

En este apartado se van a recoger las conclusiones que se han sacado como parte del trabajo y desarrollo de este PFG. Para ello se desarrollará una serie de aspectos que han sido importantes en el aprendizaje del PFG.

El PFG se basa en JavaFX 2.2 que es la última versión de esta tecnología. Al desarrollar con una tecnología novedosa se nota que no se le ha podido sacar el máximo debido a que no existen muchos sitios donde poder ver todo su potencial.

Esto unido a que mis conocimientos de patrones de diseño eran muy limitados al principio del PFG, han dado lugar a que la aplicación sea menos robusta de lo que me gustaría. A pesar de llevar unos meses mejorando la aplicación cuanto más me enfrentaba a los problemas que iban surgiendo, más aprendía de la necesidad de conocer bien la tecnología y que el diseño de la aplicación fuese muy bueno.

Una vez que he ido conociendo todos los entresijos de la aplicación y mejorando el diseño de la misma, el PFG ha ido creciendo en robustez y era más sencillo el introducir nuevas funcionalidades.

Después de desarrollar la aplicación me he dado cuenta de que un conocimiento de los patrones de diseño, es algo imprescindible para poder desarrollar aplicaciones software sea lo que sea lo que se quiere desarrollar. En la etapa de diseño los cambios son mucho más sencillos de realizar que cuando se tienen que realizar en la etapa de desarrollo. Con esto quiero decir que he de adquirir más conocimientos sobre patrones de diseño para poder aplicarlos en mi trabajo y así crecer como desarrollador y analista.

Con JavaFX me he dado cuenta de que las tecnologías cambian con mucha rapidez y que con un trabajo constante se puede dominar una tecnología. Las aplicaciones de escritorio necesitan otro tipo de mentalidad por parte del desarrollador y como dije al principio de esta memoria creo que después de terminar este PFG soy un mejor programador, aunque me doy cuenta que aun me queda mucho trabajo por delante para poder conseguir un nivel suficiente como para seguir escalando en mi carrera profesional.

9. Desarrollo futuro

En algún momento este PFG podría querer ser mejorado en cuanto a prestaciones y para ello se dan algunas ideas sobre posibles mejoras que se podrían incluir, ampliando o modificando la aplicación, en un futuro.

- **Extender la aplicación a nuevas anotaciones**: Una mejora interesante sería extender el uso de la aplicación a diferentes anotaciones permitiendo al usuario elegir entre una mayor variedad de anotaciones con las cuales enriquecer los libros.
- **Añadir búsqueda de texto en las anotaciones:** Una funcionalidad que se podría añadir seria la búsqueda en las anotaciones de palabras o frases para mejorar la aplicación actual.
- **Optimizar la aplicación**: Debido a mi inexperiencia con la tecnología JavaFX y a la programación con patrones de diseño, un importante paso sería optimizar el código para que todo lo que se ha escapado funcione mucho más rápido.
- **Crear un portal donde se puedan descargar más libros**. Un proyecto en si mismo podría ser la creación de un portal desde el cual descargar los libros que se pueden leer en este PFG. Hasta ahora estos libros han sido generados en local y son cargados desde el mismo equipo pero si se aumenta la aplicación para que el lector se conecte a un portal y se descargue de allí los libros sería una buena idea.
- **Mejorar el aspecto visual de la aplicación:** Gracias a la capacidad que tiene JavaFX para usar los ficheros CSS en la mejora de la apariencia de las aplicaciones, con un diseñador se puede hacer que la belleza de la aplicación sea mucho mayor que la que actualmente posee.

10. Bibliografía

Libros:

- Gamma, E. ; Helm,R. ; Johnson,R. y Vlissides,J. (2003). Patrones de Diseño. Elementos de software orientado a objetos reutilizable. Madrid. Pearson Educación, S.A.
- Weitzenfeld, A. (2005). Ingeniería de Software Orientada a Objetos con UML, Java e Internet. México. Thomson.
- Weaver, J.L. ; Gao, W. ; Chin, S. y Iverson, D. (2012). Pro JavaFX 2. A definitive Guide to Rich Clients with Java Technology. New York. Apress.
- Salvador Sánchez; Miguel ángel Sicilia; Daniel Rodriguez (2011) Ingeniería del Software. Un enfoque desde la guía SWEBOK, Madrid, Garceta Gupo Editorial.

Páginas Webs:

- Wikipedia (2014). Adobe Flash [Internet] Disponible en: <http://es.wikipedia.org/wiki/Adobe_Flash_Professional> Accedida en Mayo de 2014
- Wikipedia (2014). Microsoft Silverlight [Internet] Disponible en: <http://es.wikipedia.org/wiki/Microsoft_Silverlight> Accedida en Mayo de 2014
- Wikipedia (2014). JavaFX [Internet] Disponible en: <http://es.wikipedia.org/wiki/JavaFX> Accedida en Mayo de 2014
- Blog Mi granito de Java (2014). Patrón Prototype [Internet] Disponible en: <http://migranitodejava.blogspot.com.es/search/label/Prototype> Accedida en Febrero de 2014
- Blog Mi granito de Java (2014). Patrón Command [Internet] Disponible en: <http://migranitodejava.blogspot.com.es/search/label/Command> Accedida en Febrero de 2014
- Blog Mi granito de Java (2014). Patrón Factory Method [Internet] Disponible en: <http://migranitodejava.blogspot.com.es/search/label/Factory%20Method> Accedida en Febrero de 2014
- Oracle (2014). Using FXML to Create a User Interface [Internet] Disponible en: < http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm> Accedida en Octubre de 2013

- Blog aprendiendo software (2014). Ventana en JavaFX [Internet] Disponible en: <http://aprendiendo-software.blogspot.com.es/2013/03/javafx-ventana-en-javafx.html> Accedida en Junio de 2014
- Sitio oficial de jSON (2014). Introducción al JSON <http://json.org /json-es.html> Accedida en Junio de 2014