Universidad de Salamanca

Departamento de Informática y Automática

Trabajo de Fin de Grado

UIComposer: sistema para generar interfaces gráficas que se puedan importar y exportar en diferentes formatos

VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

Autor:
Mario Fernández Álvarez

Tutores:
Francisco José García Peñalvo
Oskar Barrio Ferreiro

Julio 2019

D. Francisco José García Peñalvo certifica que este documento titulado "UIComposer: sistema para generar interfaces gráficas que se puedan importar y exportar en diferentes formatos", y el resto de anexos incluidos, han sido realizados por D. Mario Fernández Álvarez, con DNI 09816232Q, y que estos constituyen la memoria del trabajo completado para superar la asignatura Trabajo de Fin de Grado del Grado en Ingeniería Informática de la Universidad de Salamanca.

En Salamanca a de Julio de 2019.

D. Francisco José García Peñalvo
Departamento de Informática y Automática
Universidad de Salamanca

Resumen

Las interfaces gráficas de usuario determinan la interacción entre persona y computadora, lo que las convierte en uno de los elementos más importantes en el desarrollo de aplicaciones software. El diseño de un interfaz gráfica puede ser determinante a la hora de garantizar la viabilidad de un producto en la era digital en la que nos encontramos. Esta importancia a nivel comercial ha generado un gran interés dentro de la industria del software en desarrollar nuevas tecnologías que introduzcan nuevos modelos y diseños que atraigan a más usuarios. Esta expansión ha traído consigo una clara mejora en la experiencia de usuario, pero también ha conllevado una mayor carga de trabajo en el diseño y desarrollo de interfaces gráficas. Esto se debe no sólo a la dificultad de desarrollar elementos interactivos más complejos, sino también al gran número de plataformas que se deben conocer en profundidad para poder dar soporte al mayor número de dispositivos posibles. Atajando este problema se consigue aumentar la productividad y reducir el tiempo empleado, algo prioritario en cualquier negocio. La herramienta UIComposer, desarrollada en este trabajo de fin de grado en colaboración conjunta de la empresa HP SCDS y la Universidad de Salamanca, ha sido creada para facilitar el diseño de interfaces gráficas de usuario. Esta herramienta ofrece la posibilidad de trabajar con distintas plataformas y tecnologías siguiendo un modelo de trabajo común. La aplicación cuenta con un editor de texto y un panel de propiedades con los que modificar el contenido de un fichero, y un visualizador que muestra los cambios realizados. La aplicación ha sido desarrollada siguiendo una arquitectura modular y extensible, la cual permite añadir nuevas funcionalidades y soporte para cualquier formato de definición de interfaces gráficas de usuario. Con esto se consigue establecer una aplicación base a la cual se le puede seguir añadiendo nuevos componentes que expandan sus capacidades.

Palabras clave: interfaces gráficas, GUI, diseño, aplicación, HTML, XAML, C#, .NET.

Abstract

Graphical user interfaces determine the interaction between person and computer, which makes them one of the most important elements in the development of software applications. The design of a graphical interface can be decisive when it comes to guaranteeing the viability of a product in the digital era in which we find ourselves. Its relevance on a commercial level has generated great interest within the software industry in developing new technologies that introduce new models and designs that attract more users. This expansion has brought along with it a clear improvement in the user experience, but it has also led to a greater workload in the design and development of graphical interfaces. This is due not only to the difficulty of developing more complex interactive elements, but also to the large number of platforms that must be thoroughly understood in order to support as many devices as possible. Tackling this problem increases productivity and reduces the time spent on it, a priority in any business. The UIComposer tool, developed in this project in joint collaboration between HP SCDS and the University of Salamanca, has been created to facilitate the design of graphical user interfaces. This tool offers the possibility of working with different platforms and technologies following a common working model. The application has a text editor and a properties panel with which the contents of a file can be modified, as well as a viewer that shows the changes that have been made. The application has been developed following a modular and extensible architecture, which allows to add new functionalities and support for any graphical user interface definition format. This establishes a base application to which new components can be added to expand its capabilities.

Keywords: graphical interfaces, GUI, design, application, HTML, XAML, C#, .NET.

Índice general

1. Introducción **1**
1.1. Estructura de la memoria 2

2. Objetivos **3**

3. Metodología y técnicas **4**
3.1. Scrum 4
3.2. DevOps 6
3.3. Gestión de la configuración 7

4. Entorno de desarrollo **9**
4.1. Plataformas y lenguajes 9
4.2. Herramientas de desarrollo 11
4.3. Herramientas de documentación 13

5. Sprints **14**
5.1. Sprint 1 15
5.2. Sprint 2 21
5.3. Sprint 3 27
5.4. Sprint 4 30
5.5. Sprint 5 36
5.6. Sprint 6 40
5.7. Sprint 7 42
5.8. Sprint 8 47

6. Producto **53**
6.1. Inicio 53
6.2. Abrir fichero 54
6.3. Fichero abierto 55
6.4. Editor de texto 56
6.5. Visualizador 57
6.6. Panel de propiedades 58

7. Conlusiones **59**

Referencias **60**

Índice de figuras

4.1. Logo .NET Core 9
4.2. Logo C# 9
4.3. Logo xUnit.net 10
4.4. Logo Visual Studio 11
4.5. Logo Git 11
4.6. Logo GitLab 12
4.7. Logo Adobe XD 12
4.8. Logo Visual Paradigm 13
4.9. Logo LaTeX 13

5.1. *Mockup* de la aplicación 17
5.2. Violación espacio reservado. 21
5.3. Plugin en la versión 0.2.0 23
5.4. Componentes de la aplicación en la versión 0.2.0 23
5.5. Carga de plugins en la versión 0.2.0 24
5.6. Carga control visual en la versión 0.2.0 25
5.7. Placas del repositorio 26
5.8. Componentes de la aplicación en la versión 0.4.3 32
5.9. Arquitectura de la aplicación en la versión 0.4.3 33
5.10. Carga control visual en la versión 0.4.3 34
5.11. Lectura de XAML en la versión 0.5.1 37
5.12. Carga de plugins en la versión 0.5.2 39
5.13. Carga fichero en la versión 0.5.2 39
5.14. Editor de texto en la versión 0.6.5 44
5.15. Panel de propiedades en la versión 0.6.6 45
5.16. Obtención de propiedades en la versión 0.6.6 46
5.17. Guardado fichero en versión 0.8.1 48
5.18. Menú archivo en la versión 0.8.3. 50
5.19. Menú edición en la versión 0.8.3. 50
5.20. Ventana acerca de UIComposer en la versión 0.8.4 51
5.21. Arquitectura final. 52

6.1. Ventana principal. 53
6.2. Abrir fichero XAML. 54
6.3. Ventana apertura fichero. 54

6.4. Ventana principal tras cargar fichero XAML. 55
6.5. Editor de texto. 56
6.6. Visualizador XAML. 57
6.7. Panel de propiedades. 58

Índice de tablas

5.1. Sprints del proyecto 14
5.2. Historia de usuario n.º 3 15
5.3. Historia de usuario n.º 4 15
5.4. Historia de usuario n.º 5 16
5.5. Historia de usuario n.º 6 18
5.6. Historia de usuario n.º 7 19
5.7. Historia de usuario n.º 8 20
5.8. Historia de usuario n.º 1 21
5.9. Historia de usuario n.º 9 22
5.10. Historia de usuario n.º 10 25
5.11. Historia de usuario n.º 11 26
5.12. Historia de usuario n.º 13 27
5.13. Historia de usuario n.º 14 28
5.14. Historia de usuario n.º 15 28
5.15. Historia de usuario n.º 18 29
5.16. Historia de usuario n.º 20 30
5.17. Historia de usuario n.º 21 31
5.18. Historia de usuario n.º 25 31
5.19. Historia de usuario n.º 22 32
5.20. Historia de usuario n.º 24[1] 35
5.21. Historia de usuario n.º 26 36
5.22. Historia de usuario n.º 28 37
5.23. Historia de usuario n.º 29 38
5.24. Historia de usuario n.º 27 40
5.25. Historia de usuario n.º 31 41
5.26. Historia de usuario n.º 32 41
5.27. Historia de usuario n.º 33 42
5.28. Historia de usuario n.º 35 43
5.29. Historia de usuario n.º 36 43
5.30. Historia de usuario n.º 37 44
5.31. Historia de usuario n.º 38 47
5.32. Historia de usuario n.º 39 49
5.33. Historia de usuario n.º 40 49
5.34. Historia de usuario n.º 41 50

1 Introducción

En los inicios de la computación la comunicación entre computador y persona se limitaba a teclear órdenes en una consola. Este sistema requiere que el usuario sea conocedor de estas instrucciones, por lo que el uso de los computadores se limitaba a los expertos. Uno de los primeros investigadores en proponer un nuevo modelo de interacción persona ordenador fue Douglas Engelbart en la que se conoce hoy en día como "La madre de todas las demos" de 1968 (Wired Staff, 2004). En esta presentación se demostraron muchos de los fundamentos de la informática moderna como las ventanas, el hipertexto, las videoconferencias o el ratón.

Con la introducción de los primeros ordenadores personales el usuario pasó a ser una persona común sin conocimientos de informática, lo que hizo necesario un cambio en el diseño de aplicaciones informáticas que facilitase su manejo con el uso de las interfaces gráficas de usuario. Uno de los primeros ordenadores comerciales que contó con una interfaz gráfica de usuario fue el Xerox Alto (Thacker, McCreight, Lampson, Sproull y Boggs, 1979), desarrollado por algunos de los colaboradores de Engelbart en Xerox PARC. Este sistema incluía muchos programas innovadores como *Bravo*, el primer editor de texto con tecnología *What You See Is What You Get* (WYSIWYG, lo que ves es lo que obtienes).

Con su interfaz gráfica de usuario basada en el ratón, el Alto influenciaría a Steve Jobs y al sistema operativo Macintosh de Apple en la década de 1980 (Gladwell, 2011). Jobs y su equipo simplificaron el diseño del escritorio permitiendo al usuario manipular directamente los elementos mostrados en pantalla. Con esto se estableció el modelo WIMP (ventanas, iconos, menús y puntero) en la interacción persona-ordenador, el cual sigue prevaleciente en los ordenadores de escritorio.

Desde entonces las interfaces gráficas de usuario han seguido evolucionando y adaptándose a las nuevas tecnologías, como los dispositivos móviles con pantallas táctiles o la realidad virtual, introduciendo nuevas formas de interactuar con una computadora. Estas nuevas tendencias han influido también en las interfaces de usuario de los ordenadores de sobremesa, aunque sigue habiendo una preferencia por la interfaz de usuario de escritorio más tradicional.

Las interfaces gráficas se han convertido en la parte fundamental en la comunicación entre usuario y computadora. Su alcance ya no se limita solo ha presentar las funciones de una aplicación informática, si no también ha proporcionar una experiencia satisfactoria al usuario. El diseño de la interfaz puede ser determinante en la aceptación de un producto digital hoy en día (Gutierrez Miranda, 2011). Debido a su importancia la industria de la informática ha invertido grandes recursos en su desarrollo, lo que ha dado lugar a un gran número de sistemas distintos.

Actualmente existe un gran número de lenguajes distintos para desarrollar interfaces gráficas. Esto se debe a que las grandes empresas tecnológicas tienden a desarrollar sus propios sistemas. Por ejemplo, Microsoft cuenta con WPF y UWP, Apple con Cocoa y SwiftUI, Google con Angular y Flutter, etc. Cada uno de ellas se implementa en lenguajes distintos, lo que produce un ecosistema fragmentado en el que resulta difícil trabajar de manera uniforme con todos ellos. Esto supone un perjuicio claro si, por ejemplo, se

pretende crear una aplicación multiplataforma, ya que no sólo implica tener que aprender un nuevo formato de definición de GUIs, si no también el manejo de la herramienta que permita trabajar con ese formato concreto.

Este proyecto surge con la idea de construir una única herramienta que permita trabajar con varios formatos existentes para la definición de interfaces gráficas de usuario. Se trabajará con dos formatos en concreto: HTML y XAML. El primero de ellos es hoy en día uno de los más usados ya no sólo para páginas web, sino también para aplicaciones de escritorio. HTML se ha convertido en un lenguaje ubicuo presente en todas las plataformas, por lo que contar con una herramienta que permita editar un fichero de este tipo resultaría de gran ayuda para aquellos que no sean conocedores del diseño de páginas web. El segundo es un formato diseñado por Microsoft para su uso en las aplicaciones de escritorio en Windows. Con él se pretende reemplazar gradualmente a las aplicaciones de Win32, introduciendo un diseño más moderno con las plataformas de Windows Presentation Foundation (WPF) y Universal Windows Platform (UWP).

1.1 Estructura de la memoria

Este documento se divide en 7 apartados en los que se detallará el trabajo realizado para desarrollar la aplicación UIComposer.

1. **Introducción**: se presentan los antecedentes históricos y la motivación de este trabajo.
2. **Objetivos**: se establecerán los objetivos y el alcance del proyecto.
3. **Metodología y técnicas**: se comentará la metodología seguida y las técnicas de ingeniería del software empleadas.
4. **Metodología y técnicas**: se enumerarán las herramientas del entorno de de desarrollo utilizadas.
5. **Sprints**: se relatará el desarrollo y la evolución de los distintos componentes de la aplicación a lo largo del proyecto.
6. **Producto**: se mostrará el producto obtenido tras finalizar el desarrollo.
7. **Conclusiones**: se expondrán las conclusiones finales y posibles mejoras futuras.

Junto con este documento se adjuntan varios anexos con documentación técnica:

I **Especificación de requisitos**: incluye los requisitos del sistema *software* siguiendo el estándar IEEE-830 (IEEE, 1998).

II **Modelo de análisis**: contiene diagramas UML en los que se muestra las relaciones e interacciones entre las clases del sistema.

III **Modelo de diseño**: explicación de la implementación y la arquitectura del sistema.

IV **Documentación de referencia**: documentación sobre las clases e interfaces de la aplicación y los plugins.

V **Manual de usuario**: guía de uso del sistema orientada al usuario final.

2 Objetivos

Inicialmente se plantean dos posibles enfoques para el proyecto. El primero consiste en una aplicación de consola capaz de transformar un fichero de un formato a otro, y el segundo, por el que se optó finalmente para este trabajo, se trata de una aplicación de escritorio que permita visualizar y editar interfaces gráficas de usuario definidas en distintos formatos.

Por tanto, el objetivo principal de este trabajo consiste en desarrollar una herramienta que permita editar ficheros de interfaces gráficas en distintos formatos, en concreto XAML y HTML. La aplicación UIComposer debe permitir trabajar de manera homogénea y accesible con distintos formatos para facilitar el trabajo y el aprendizaje de diseño de GUIs. Para cumplir con este objetivo la herramienta debe contar los siguientes elementos:

- Un editor de texto en el que se mostrará el contenido del fichero abierto. Con este editor se podrá modificar el contenido del fichero directamente a través del teclado, y la aplicación permitirá guardar dichos cambios en el fichero original o en otro nuevo.
- Un visualizador con el que poder ver la interfaz definida en el fichero. El visualizador deberá también actualizarse según los cambios realizados sobre el fichero original.
- Un listado con las propiedades del elemento seleccionado del la interfaz gráfica. Se presentará el nombre y el valor de la propiedad, el cual podrá ser modificado. Estos cambios también deberán reflejarse en el editor de texto y en el visualizador.

Con esto la herramienta UIComposer proporcionará un servicio completo con el que poder editar las interfaces gráficas definidas en el fichero cargado, ya sea directamente con el editor o con las propiedades mostradas, y visualizar los cambios aplicados para comprobar su efecto final.

Además de este objetivo principal con el que se define la funcionalidad de la aplicación se incluye otro en cuanto al diseño arquitectónico de la aplicación. Su arquitectura tiene que ser modular y extensible, de modo que los componentes necesarios para añadir soporte a un nuevo formato puedan ser definidos aparte de la aplicación principal. Este objetivo se fija no solo para establecer una separación clara entre el desarrollo de la aplicación principal y el de los componentes externos, sino también para ofrecer la posibilidad de expandir la funcionalidad de la aplicación en el futuro.

Por último, la integración y el despliegue de la aplicación deben ser un proceso automatizado. Primero, la integración de la aplicación mediante la compilación y ejecución de pruebas tiene que ser transparente, para así poder detectar posibles errores en el desarrollo. Con esto se previene que los errores no detectados se propaguen y provoquen un daño mayor, pudiendo resolverlos a tiempo. Después el despliegue debe ponerla a disposición de cualquier posible usuario automáticamente, realizando las operaciones necesarias de compilación y de empaquetamiento de todos los componentes.

3 Metodología y técnicas

Dentro del desarrollo del software se pueden seguir distintos tipos de procedimientos para mejorar la calidad del diseño y la gestión del proyecto (García-Peñalvo y Vázquez-Ingelmo, 2019). En este caso se ha optado por seguir una metodología ágil que permita trabajar siguiendo un modelo iterativo e incremental. En el Manifiesto por el Desarrollo Ágil de *software* (Beck et al., 2001) se establecen 12 principios en los que se priorizan la adaptabilidad ante los nuevos requisitos, la eficiencia en la comunicación, la calidad del *software* y el progreso del producto al final de cada iteración.

Además, para cumplir con los objetivos de integración y despliegue automáticos, se aplicarán en conjunción con la metodología ágil las técnicas de integración continua y despliegue continuo, que forman parte de lo que se conoce como DevOps. Este tipo de prácticas sirven para acelerar el proceso de validación de cambios y de entrega del producto, con lo que se requiere menos trabajo en estas áreas.

Por último, se comentará la gestión de la configuración, especificando los procesos de identificación y control de cambios. Estas tareas son importantes teniendo en cuenta que en las metodologías ágiles las cambios se producen frecuentemente, por lo que se debe mantener un seguimiento apropiado.

3.1 Scrum

Scrum (Schwaber, 2004) es un marco de trabajo donde se definen una serie de prácticas para favorecer el trabajo en equipo. Scrum se incluye dentro de las metodologías ágiles ya que cumple con sus principios básicos, permitiendo una planificación y organización del trabajo flexible.

3.1.1 Roles

En Scrum se describen una serie de roles para cada uno de los participantes:

- El **dueño del producto**, responsable de definir los requisitos y de priorizar las tareas.
- El **equipo de desarrollo**, encargado de entregar un producto al final de cada iteración.
- El ***Scrum Master***, que debe asegurarse de que se siga las prácticas definidas en Scrum.

Dado que este trabajo no se trata de un proyecto colaborativo con múltiples participantes es necesario adaptar estos roles para este contexto. Se podría decir que yo realizaré el papel de los tres, aunque también hay que tener en cuenta la aportación de los tutores en el desarrollo del proyecto. Por tanto, el desarrollo del *software* será mi responsabilidad, y el seguimiento de la planificación y las prácticas de Scrum será una tarea conjunta.

3.1.2 Sprint

El elemento central dentro de la planificación en Scrum son los Sprints, un periodo de tiempo de entre 1 a 4 semanas durante el cual se lleva a cabo el trabajo. El Sprint comienza con la planificación, una reunión inicial donde los participantes establecen los objetivos a cumplir durante el mismo y las tareas a llevar a cabo para cumplirlos. Durante el desarrollo del Sprint también se realizarán reuniones diarias para que el equipo de desarrollo se coordine. Al finalizar el Sprint los participantes se reúnen de nuevo para la revisión y la retrospectiva del Sprint. En la primera se revisan las tareas completadas y se realiza una demostración del producto obtenido como resultado. En la segunda se evalúa el Sprint y se proponen mejoras sobre la planificación.

En este proyecto los Sprints serán de 3 semanas, y las reuniones de planificación, revisión y retrospectiva se realizarán los Lunes por videoconferencia con el tutor de HP SCDS. El tiempo total será de 1 hora como máximo, durante la cual se revisarán las tareas completadas, se realizará una demostración de la aplicación, se evaluará el progreso del trabajo y se establecerán las tareas para el siguiente Sprint.

3.1.3 Historias de usuario

Una historia de usuario es una descripción de una característica del sistema *software* desde la perspectiva del usuario. Representa la unidad básica de trabajo a realizar en el Sprint, y se incluyen ordenadas por prioridad en el *Backlog*. Las historias de usuario en el trabajo seguirán la siguiente plantilla:

COMO usuario *QUIERO* necesidad que quiero cubrir *ENTONCES* qué es lo que tengo que hacer para lograrlo

Siguiendo este modelo para las historias de usuario se establecen claramente los requisitos a cumplir y las actividades a completar para conseguirlo. En cada historia de usuario se incluirán también todas las subtareas necesarias para completarla, y unos criterios de aceptación con los que se pueda determinar si la tarea se ha completado, lo que se conoce como la definición de terminado. Además, cada historia de usuario deberá seguir los principios INVEST (Hartman, 2009):

- **I**ndependiente del resto de tareas.
- **N**egociable, resultado de la colaboración de todo el equipo.
- Con un **v**alor claro que permita ordenarlas por prioridad.
- **E**stimable en cuanto al tiempo que requiere completarla.
- **P**equeña, de corta duración.
- **T**estable, que permita comprobar claramente si se ha completado o no.

3.1.4 Spikes

Los *spikes* son actividades que no se suelen definir como historias de usuario, si no que se usan para investigar de antemano algún elemento del que no se tiene mucho conocimiento. Su propósito es obtener el conocimiento necesario para comprender mejor un requisito o aumentar la fiabilidad de una estimación de una historia de usuario. En este proyecto se

utilizarán generalmente como un paso previo a la implementación de un nuevo componente, para explorar las posibles opciones y valorar cual resulta ser la más óptima.

3.2 DevOps

Es difícil encontrar una definición formal aceptada al tratarse de un ámbito de trabajo explorado recientemente, pero se puede decir que DevOps es un conjunto de prácticas de ingeniería del *software* con las que se combina el trabajo de desarrollo (*Dev*) con el de operaciones (*Ops*), eliminando la barrera existente entre los que desarrollan el *software* y los que lo operan (Loukides, 2017). Estas prácticas surgieron a raíz del gran crecimiento de la infraestructura de las páginas web, cuyo desarrollo se caracteriza por los cambios frecuentes. Al aumentar el número de desarrolladores y recursos la estabilidad se reduce, lo cual supone una pérdida mayor debido al crecimiento del número de usuarios. Las prácticas de DevOps comienzan entonces a aplicarse con el objetivo de conseguir mayor escalabilidad y estabilidad.

Si bien este proyecto no es del mismo calibre que las grandes páginas web o los servicios en la nube, también se puede beneficiar de estas prácticas al aplicarlas en conjunto con Scrum. Al final de los ciclos de desarrollo cortos de las metodologías ágiles se debe conseguir un producto entregable, lo cual supone un incremento en el trabajo de operaciones. Aplicando las prácticas de DevOps este se reduce y se consigue un modelo de trabajo automatizado y transparente (Santos Luaces, 2019).

Dentro de DevOps se pueden encontrar una gran variedad de técnicas distintas orientadas a distintos tipos de servicios, como la monitorización o la conteinarización. Dado que este trabajo se limitará al ámbito académico no es necesario aplicar este tipo de prácticas destinadas al *software* a nivel comercial. Por tanto, dados los objetivos establecidos, las dos prácticas principales a aplicar serán la integración continua y el despliegue continuo.

3.2.1 Integración continua

La integración continua, conocida comúnmente por sus siglas del inglés CI (*continuous integration*, consiste en integrar automáticamente un proyecto *software*, es decir, comprobar que la compilación y las pruebas se ejecutan correctamente. Esto permite detectar los posibles fallos provocados por las nuevas modificaciones añadidas al proyecto, evitando así que estos errores lleguen al producto final.

Para poder llevar a cabo esta práctica se requiere un repositorio central donde se aloje el código de la aplicación, y un entorno donde se realicen las tareas de complicación y pruebas automáticamente. Estos se verán más adelante en el apartado 4.2.3

3.2.2 Despliegue continuo

El despliegue continuo es una estrategia del desarrollo *software* en la que cualquier cambio que haya pasado correctamente por las pruebas de integración continua se pone a disposición del usuario final automáticamente. Esta estrategia se suele confundir con la entrega continua ya que ambos comparten las mismas siglas del inglés, CD, pero la diferencia clave es la automatización. En el despliegue continuo una vez se hayan superado

satisfactoriamente las pruebas establecidas el nuevo cambio se aplica al producto final y se despliega. En cambio, en la entrega continua este último paso es manual, ya que requiere que una persona apruebe los cambios realizados para asegurarse de su calidad.

El despliegue continuo es una práctica eficiente siempre y cuando se establezcan unos criterios claros para la aceptación de una cambio. Es por ello que resulta apropiado para este trabajo dado que en cada historia de usuario se incluye la definición de terminado. Con esto se consigue agilizar la gestión de la configuración de los distintos elementos del sistema, como se verá en el siguiente apartado.

3.3 Gestión de la configuración

La gestión de la configuración es una disciplina de la ingeniería del *software* que sirve para controlar los cambios realizados en un sistema *software*. Su propósito es establecer y mantener la integridad del producto durante el ciclo de vida del desarrollo. El plan de gestión de la configuración establecido para este proyecto se centrará en el control de versiones y cambios de los elementos de configuración *software*. Las distintas actividades del plan se realizarán acorde al calendario establecido, en paralelo a las actividades de desarrollo y documentación, y realizando las revisiones pertinentes al finalizar cada Sprint.

3.3.1 Control de versiones

El control de versiones se limitará al código fuente implementado para la aplicación, y se realizará siguiendo la nomenclatura del versionado semántico (Preston-Werner, 2013). En este estándar las versiones se identifican únicamente con tres números positivos X (*major*), Y (*minor*), y Z (*patch*) de la forma "X.Y.Z". En este caso se seguirá la misma definición para X y Z, solo cambiará Y:

- X se incrementará cuando se realice un cambio mayor que haga que la API sea incompatible con la versión anterior.
- Y indicará el Sprint de la planificación temporal.
- Z se incrementará cuando se corrija algún error o se añada alguna nueva funcionalidad compatible.

Al incluir en Y el número del Sprint de la versión resulta más fácil identificar las distintas versiones obtenidas al final de cada uno. La primera versión de la aplicación será la "0.1.0", y se incrementará a medida que se completen las tareas y los Sprints.

Para poder llevar a cabo este proceso de manera automática se utilizarán las herramientas git y GitVersion como se verá en el apartado subsec:gitversion.

3.3.2 Control de cambios

Los cambios se implementarán según los requisitos de las historias de usuario en una nueva rama de desarrollo del repositorio central. Para que un cambio sea aprobado deberá cumplir los requisitos establecidos, superar las pruebas unitarias para los cambios implementados, y documentar apropiadamente todo el proceso.

Una vez se haya aprobado se incluirá en la configuración de referencia, que en este

caso se corresponde con la rama principal del repositorio. Usando la herramienta git se realizará una petición *merge* de la rama creada con la rama principal, resolviendo todos los posibles conflictos que puedan aparecer. Tras esto se podrá dar por finalizada la tarea correspondiente.

Cuando se desee realizar algún cambio sobre un elemento de configuración que haya sido añadido a la configuración de referencia, este deberá pasar por el mismo proceso, es decir, definir la historia de usuario, implementar los cambios y evaluar si cumplen criterios de aprobación establecidos.

4 Entorno de desarrollo

El entorno de desarrollo está formado por el conjunto de herramientas y procesos empleados para crear el producto software. Para este proyecto se contará con todas las herramientas, servicios y procesos necesarios para poder llevar a cabo la implementación, las pruebas, el despliegue y la documentación. En los siguientes apartados se detallarán las plataformas y lenguajes utilizados para implementar la aplicación UIComposer, las herramientas empleadas para el desarrollo del propio software, y por último las herramientas con las que se ha generado esta documentación. También hay que destacar de antemano que el sistema operativo usado durante el desarrollo será Windows, concretamente la última versión Windows 10.

4.1 Plataformas y lenguajes

4.1.1 .NET Core 3

Fig. 4.1. Logo .NET Core

Hasta ahora para el desarrollo de aplicaciones de escritorio en Windows con .NET solo se podía utilizar .NET Framework, pero con la versión 3 de .NET Core se añadió el soporte para GUIs usando Windows Forms, WPF o UWP. Debido a esto .NET Core tiene ahora mayor preferencia para las aplicaciones de escritorio en Windows, y teniendo en cuenta también que su diseño es más moderno y ligero que su antecesora, .NET Core 3 resulta ser la mejor opción para la aplicación. El único incoveniente que se puede plantear es que a día de hoy aún se encuentra en fase de desarrollo, por lo que algunas de sus características no son estables.

4.1.2 C# 8

Fig. 4.2. Logo C#

C# es un lenguaje de programación orientado a objetos desarrollado por Microsoft como parte de la plataforma .NET. Una de las características más importantes de este lenguaje es que al compilarlo no se genera código máquina nativo, sino que se convierte a código CIL (*Common Intermediate Language*), el cual es compilado durante la ejecución del programa a código máquina por el CLR (*Common Language Runtime*). Se utilizará la versión 8 de este lenguaje que introduce nuevas características como las declaraciones *using* que se usarán el proyecto.

4.1.3 | Windows Presentation Foundation

Puesto que la plataforma escogida es .NET las opciones para la implementación de la interfaz de la aplicación se limitan a las compatibles con ella, que son *Windows Forms*, *Windows Presentation Foundation* (WPF) y *Universal Windows Platform* (UWP). De entre ella se ha escogido a WPF porque es más moderna que Windows Forms, pero también es compatible con las versiones anteriores de Windows a diferencia de UWP.

WPF es una tecnología desarrollada por Microsoft para crear interfaces de usuario enriquecidas y sofisticadas para Windows. WPF emplea XAML, un lenguaje basado en XML, para definir y enlazar varios elementos de la interfaz. En WPF, XAML forma un lenguaje de marcado de la interfaz de usuario para definir elementos de interfaz de usuario, vinculación de datos, eventos y otras características. Los elementos XAML se asignan directamente a los objetos de CLR, mientras que los atributos XAML se asocian a las propiedades y eventos en esos objetos.

4.1.4 | xUnit.net

Fig. 4.3. Logo xUnit.net

xUnit.net es una plataforma para la implementación de pruebas unitarias en .NET. Forma parte del conjunto de herramientas conocido como xUnit, que siguen el diseño establecido inicialmente en SUnit (Beck, 1989) en el que se definen varios elementos:

- *Test fixture*, que contiene las precondiciones para las pruebas.
- *Test suite*, un conjunto de pruebas que comparten el mismo fixture.
- Método setup(), en el que se realizan los cambios necesarios al entorno de ejecución para poder realizar una prueba.
- Método teardown(), en el que se deshacen los cambios anteriores.

En base a este patrón de diseño se pueden distinguir tres diferencias claves entre xUnit.net y el resto de plataformas para pruebas unitarias en .NET como MSTest o NUnit:

- No se utiliza ningún atributo especial para indicar una clase como *fixture* si no que se buscan los métodos de prueba en todas las clases públicas.
- En lugar de setup() se utiliza el constructor de la clase donde se encuentran las pruebas para establecer el contexto de las pruebas.
- Tampoco se utiliza teardown(), en su lugar se utiliza la interfaz "IDispose" para desechar los recursos usados.

Como se puede comprobar el beneficio principal xUnit.net es que resulta más sencillo y ligero a la hora de definir las pruebas de usuario. Además estas pueden ser ejecutadas también usando la línea de órdenes, por lo que se pueden añadir directamente al proceso de integración continua.

4.2 Herramientas de desarrollo

4.2.1 Visual Studio 2019

Fig. 4.4. Logo Visual Studio

Visual Studio es un entorno de desarrollo integrado (IDE) con soporte para un gran número de lenguajes de programación como C++, C#, Java o Python. Incluye utilidades como *IntelliSense* para el autocompletado, depurador y diseñador de GUI, y permite añadir otras mediante extensiones.

Se ha optado por este IDE debido a la experiencia previa con él y a que es el más completo para trabajar con la plataforma .NET al ser desarrollado también por Microsoft. Inicialmente se trabajó con la versión de 2017, pero por problemas de incompatibilidad con .NET Core 3 finalmente se empleará la versión *Enterprise* de este año 2019 con soporte para .NET Core 3 y C# 8. Esta versión incluye características como *IntelliTrace* y *IntelliTest* que resultarán muy útiles en la depuración y en las pruebas, respectivamente.

4.2.2 Git

Fig. 4.5. Logo Git

Git es un sistema distribuido de control de versiones que permite gestionar los cambios realizados en un fichero. Su funcionamiento se basa en tomar una instantánea (copia) del fichero con cada cambio que se guarda en el repositorio. Su otra característica fundamental son las ramas, las cuales crean una bifurcación en el registro del repositorio que permite trabajar en paralelo y realizar cambios sin que afecte a la otra rama. Cuando se finaliza el trabajo en la nueva rama se puede unir de nuevo a la rama anterior, aplicando todos los cambios que se haya recogido.

Para este trabajo como norma general se creará una nueva rama por cada nueva tarea del backlog que se empiece, y cuando esta finalice se unirá de nuevo con la rama principal. Otra de las normas del repositorio será su organización, que se dividirá en tres directorios:

- La carpeta *code* contendrá el código fuente de la aplicación. Esto incluye la solución de Visual Studio y sus proyectos, y también otros *scripts* utilizados en el proceso de despliegue continuo.
- En la carpeta *docs* se incluirán otros recursos del proyecto de pequeño tamaño (máximo 30MB). Cualquier otro recurso más grande se guardará en un servicio en la nube y se referenciará en la wiki del repositorio.
- *spikes* almacenará los spikes de investigación que se realicen. Los *spikes* son también proyectos de Visual Studio, pero conviene mantenerlos por separado ya que no forman parte de la aplicación en sí.

4.2.3 | GitLab

Fig. 4.6. Logo GitLab

GitLab es un servicio web basado en Git que se utilizará como repositorio remoto del proyecto, de modo que los cambios realizados localmente se subirán al repositorio para que puedan ser propagados al resto de participantes del sistema distribuido. Además del control de versiones a través de Git este servicio cuenta con muchas otras características que se utilizarán en este trabajo, concretamente definir las historias de usuario siguiendo la plantilla establecida, mantener el backlog del producto priorizando las tareas, planificar el calendario de trabajo en los Sprints que se marcarán como hitos, alojar la documentación del proyecto en una wiki, y las actividades de integración y despliegue continuos.

El uso de este servicio ha sido establecido por la empresa HP-SCDS para poder consultar los cambios realizados. La administración del repositorio correrá a cargo de esta entidad, y el alumno podrá subir los cambios que haya realizado, crear nuevas historias de usuario e hitos desarrollo, y gestionar el sistema de CI/CD.

4.2.4 | GitVersion

GitVersion es una herramienta que trabaja directamente con el repositorio git del proyecto para automatizar la identificación de las versiones del software. Al ejecutar la herramienta dentro del repositorio analizará las etiquetas presentes y asignará a los proyectos creados con Visual Studio el número de versión indicado en las etiquetas, el cual seguirá el ya mencionado versionado semántico.

Este proceso será integrado dentro de el despliegue continuo, de modo que cuando se publique la aplicación el producto final será identificable con el número de versión. Este se incluirá tanto en las propiedades los ficheros ejecutables y de librería de la aplicación como en la propia interfaz de la aplicación.

4.2.5 | Adobe XD

Fig. 4.7. Logo Adobe XD

Adobe XD es un editor gráfico usado para diseñar y prototipar interfaces gráficas de usuario. Sirve para el diseño vectorial, el wireframing de sitios web, y la creación de prototipos interactivos y sencillos. En comparación con otras herramientas de este estilo cabe destacar su facilidad de uso y la capacidad de crear animaciones automáticamente, lo que añade la posibilidad de mostrar la interacción deseada en el prototipo de la interfaz.

En el proyecto esta herramienta se utilizará para crear un *MockUp* en el que se incluirá el diseño preliminar de la aplicación, con los componentes principales y los colores elegidos. También se trazará la interacción básica con la interfaz de la aplicación usando animaciones para simular las acciones del usuario.

4.3 Herramientas de documentación

4.3.1 Visual Paradigm

Fig. 4.8. Logo Visual Paradigm

Visual Paradigm es una herramienta CASE (*Computer-Aided Software Engineering*) que permite crear diagramas de distintos lenguajes de modelado. Esta herramienta ha sido utilizada en varias asignaturas de la titulación de Ingeniería Informática para crear diagramas usando el Lenguaje Unificado de Modelado (UML). UML es un lenguaje gráfico para visualizar, especificar y documentar un sistema de *software*. UML ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales como funciones del sistema, y aspectos concretos como componentes de software reutilizables.

Para documentar este trabajo se utilizarán diagramas de UML de varios tipos:

- **Diagrama de clase**: describe la estructura de un sistema mostrando las clases del sistema, sus atributos, métodos y las relaciones entre objetos.
- **Diagrama de componentes**: muestra las relaciones estructurales entre los componentes de un sistema.
- **Diagrama de secuencia**: usado para mostrar las interacciones entre objetos en el orden secuencial en que ocurren esas interacciones.

4.3.2 Doxygen

Doxygen es una herramienta que permite generar documentación automáticamente a partir de los comentarios en los ficheros fuente del proyecto. Gracias a ella se generará documentación en formato HTML con una navegación sencilla. Los comentarios seguirán un formato XML específico (Microsoft, 2017a), y se agregarán encima de la definición de un tipo o un miembro. Los proyectos de Visual Studio han sido configurados para que exporten automáticamente los comentarios a un fichero con formato XML, que podrá ser analizados por Doxygen.

4.3.3 LaTeX

Fig. 4.9. Logo LaTeX

LaTeX es un sistema de maquetación de alta calidad que incluye características diseñadas para la producción de documentación técnica y científica. Al utilizar este sistema frente a otras herramientas ofimáticas se obtiene un resultado de mayor calidad, por lo que esta memoria y los anexos adjuntos serán producidos con él para conseguir una mejor presentación. Para trabajar con este sistema en Windows se utilizará la distribución *MiKTeX* y el editor *Texmaker*.

5 Sprints

En esta sección se podrá ver la evolución del trabajo a lo largo de los Sprints planificados. En el apartado 3.1.2 se comentó ya que su longitud será de 3 semanas, con una reunión los lunes para revisar el trabajo realizado a lo largo de ese periodo y para planificar el siguiente Sprint. El calendario del proyecto se dividirá por tanto en 8 sprints, comenzando a finales de enero de 2019 con el comienzo del segundo cuatrimestre de la titulación de Ingeniería Informática, y acabando a finales de junio del mismo año cerca del plazo de entrega del Trabajo de Fin de Grado.

Sprint	Inicio	Fin	Objetivos
1	21/01/2019	08/02/2019	Asentamiento de la infraestructura necesaria para comenzar con el desarrollo.
2	11/02/2019	01/03/2019	Despliegue continuo, definición de la arquitectura extensible de la aplicación e implementación de plugin HTML.
3	04/03/2019	25/03/2019	Creación de plugin XAML para ficheros sencillos.
4	25/03/2019	15/04/2019	Resolución de errores en los plugins XAML y HTML y actualización de la documentación.
5	15/04/2019	06/05/2019	Solución de los fallos en los procesos CI/CD y mejora del plugin XAML para ficheros complejos.
6	06/05/2019	27/05/2019	Implementación del panel de propiedades.
7	27/05/2019	17/06/2019	Completar el desarrollo del panel de propiedades.
8	17/06/2019	28/06/2019	Añadir función de guardado de ficheros.

Tabla 5.1: Sprints del proyecto

A continuación se expondrán los Sprints completados incluyendo las historias de usuario realizadas en cada uno de ellos. La numeración indicada para las historias de usuario se corresponde con la del repositorio en GitLab.

5.1 Sprint 1

El objetivo de este Sprint será el de establecer la infraestructura básica para poder comenzar con el desarrollo de la aplicación. El Sprint comenzó el 21/01/2019 y finalizó el 08/02/2019. Las historias de usuario realizadas fueron las siguientes:

Subir spike .NET Core a repositorio	
Descripción	*COMO* tutor del proyecto *QUIERO* que todo el código generado esté disponible para consultarlo en cualquier momento *ENTONCES* necesito que el primer spike implementado sea subido al repositorio de Git
Tareas	1. Subir el proyecto y ficheros de código asociados al spike a la carpeta spikes del repositorio. 2. Documentación. 3. Pull request.
Criterios de aceptación	Todos los ficheros del spike deben de estar contenidos en una carpeta con un nombre descriptivo.

Tabla 5.2: Historia de usuario n.º 3

Antes de la reunión inicial de este primer Sprint se implementó un spike para probar el funcionamiento de la plataforma .NET Core 3. Con esta prueba se pudieron comprobar las modificaciones necesarias a aplicar al entorno de desarrollo para poder trabajar con dicha plataforma. En concreto fue necesario descargar el kit de desarrollo software de .NET Core 3 y configurar Visual Studio 2017 para usarlo.

Documentar SCM	
Descripción	*COMO* desarrollador *QUIERO* conocer exactamente que entorno de desarrollo y aplicaciones necesido para trabajar en la solución *ENTONCES* necesito que documentar en la Wiki todos los pasos y herramientas necesarias para poner en funcionamiento el entorno de desarrollo del proyecto, como el IDE de compilación, el programa de mock ups, donde se documenta, etc.
Tareas	1. Crear una nueva página en la Wiki dentro de SCM. 2. Documentar el entorno de desarrollo y tools asociadas. 3. Documentar los procesos definidos.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Todo el software necesario debe de estar referenciado. 3. Los criterios definidos para SCRUM deben de estar referenciados.

Tabla 5.3: Historia de usuario n.º 4

Con esta actividad se establecieron las herramientas y configuraciones necesarias para poder llevar a cabo el proyecto que aparecen en el apartado 4, y los procesos de gestión de la configuración descritos en el apartado 3.3.

Implementar un Mockup básico de la UI	
Descripción	*COMO* desarrollador *QUIERO* tener una referencia clara de la UI que debo implementar para la aplicación *ENTONCES* necesito que la UI esté definida a nivel visual y funcional (básico) antes de comenzar a implementarla.
Tareas	1. Crear un nuevo fichero de mockup en "docs\mockups". 2. Definir el layout general de la aplicación. 3. Documentar los procesos definidos. 4. Definir el comportamiento básico de la aplicación. 5. Documentación. 6. Pull request.
Criterios de aceptación	1. El fichero de mockup debe de estar referenciado en la Wiki. 2. El mockup debe de reflejar las tres columnas definidas para el layout básico. 3. El mockup debe de reflejar al menos cómo se abriran los ficheros externos.

Tabla 5.4: Historia de usuario n.º 5

Un *Mockup* de una interfaz de usuario permite mostrar el diseño final sin llegar a implementar el software. En el *Mockup* se refleja la paleta de colores elegida, la distribución de los elementos y la interacción básica de navegación, aunque su grado de complejidad puede variar. Para realizarlo se exploraron varias opciones, pero finalmente se optó por la aplicación Adobe XD comentada en la sección 4.2.5.

El *Mockup* de la aplicación se ha diseñado de acuerdo a la idea inicial sobre la interfaz, según la cual la vista principal de la aplicación se divide en tres columnas redimensionables como se muestra en la figura 5.1.

- En la columna izquierda se mostrará el contenido del fichero seleccionado, pudiendo seleccionar los distintos controles visuales para editarlos manualmente o desde la columna con las propiedades.
- En la columna central se visualizará la interfaz correspondiente al fichero abierto, permitiéndose ampliar o disminuir el tamaño de la vista arrastrando un control situado en la esquina inferior derecha.
- En la columna derecha aparecerán las distintas propiedades del elemento seleccionado, agrupadas en categorías para conseguir una navegación estructurada.
- En la parte superior habrá una barra que incluirá varios botones con los que se desplegarán submenús con distintas acciones, como abrir un archivo, guardar el archivo actual o salir de la aplicación.

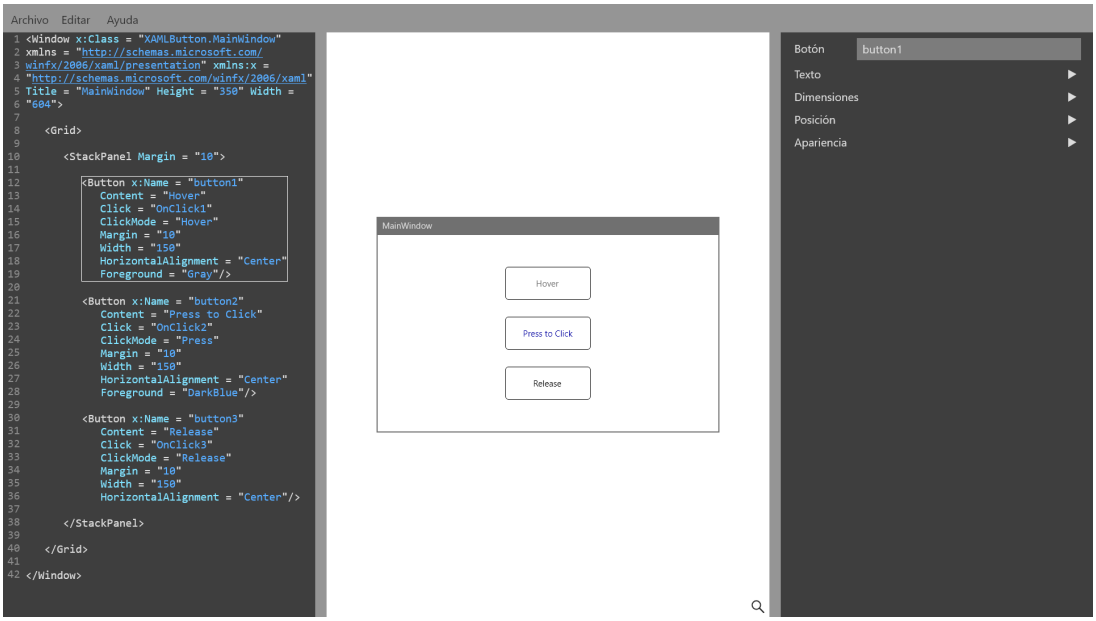

Fig. 5.1. *Mockup* de la aplicación

Los colores han sido escogidos para cumplir como mínimo con la calificación AA de la guía de accesibiliad WCAG (Caldwell, Cooper, Guarino Reid y Vanderheiden, 2008), de modo que los colores del texto y del fondo tienen un alto contraste para que el texto sea legible. El color de fondo de las columnas izquierda y derecha serán de un tono gris oscuro (■ #3E3E3E), y el de la barra superior y los menús será un tono más claro (■ #959595). Para mantener el contraste, el color del texto en las columnas laterales será más claro (■ #DCDCDCJ) y en el menú más oscuro (■ #2E2E2E), aunque cabe la posibilidad de que en la aplicación final se permita cambiar los colores para el código mostrado en la columna izquierda.

El *Mockup* incluye animaciones y transiciones con las que se muestra la interacción básica con algunos de los elementos de la interfaz, como el menú de la barra superior, las guías para redimensionar las columnas y las propiedades de la columna derecha. Estas interacciones se pueden ver en el vídeo de demostración disponible en este enlace: https://drive.google.com/file/d/1Ry5C9dUkiohoFqqt14yBCStF9Lpm1j2Q.

Crear solución básica en Visual Studio	
Descripción	*COMO* desarrollador *QUIERO* tener un espacio de trabajo para implementar el código del proyecto *ENTONCES* necesito disponer de una solución básica de Visual Studio y los proyectos básicos de referencia.
Tareas	1. Crear la solución de Visual Studio para proyectos .NET Core. 2. Crear un proyecto de tipo librería para implementar la lógica. 3. Crear un proyecto de tipo WPF para .NET Core para implementar la capa de presentation. 4. Crear un proyecto de tipo Unit testing. 5. Documentación. 6. Pull request.
Criterios de aceptación	1. Los resultados/documentación deben de estar en la Wiki. 2. La solución de VS debe de compilar para todas las plataformas. 3. Todos los ficheros asociados deben de generarse dentro de la carpeta code del repositorio.

Tabla 5.5: Historia de usuario n.º 6

La herramienta Visual Studio permite organizar los distintos ficheros con código fuente o recursos en lo que se denomina un proyecto. Un proyecto se define en un archivo XML con extensión .csproj que contiene una estructura jerárquica de carpetas virtuales y rutas a todos los elementos del proyecto. Los proyectos están contenidos a su vez dentro de una solución, donde se agrupan uno o más proyectos que forman conjuntamente la aplicación final. En el fichero de texto con extensión .sln de la solución se incluyen todos los proyectos que forman parte de ella y las relaciones de dependencia que existen entre ellos.

En este caso la aplicación se dividirá en dos proyectos, uno contendrá la lógica y el otro la interfaz de usuario. El primero será un proyecto llamado *Lib* de tipo de librería, con lo cual se generará un archivo con extensión .dll al compilarlo. El segundo será un proyecto llamado *View* de tipo WPF que será fijado como proyecto de inicio, de modo que cuando se compile se generará un ejecutable al cual se podrá acceder durante la depuración. También tendrá una relación de dependencia con el primero, lo que permitirá acceder a las clases, interfaces y métodos definidos en él.

Por último se creará un proyecto donde se definirán las pruebas unitarias, con las que se se comprueban componentes y métodos de software individuales. Esto sirve para asegurar que cada unidad funciona correctamente por separado y aislar los posibles errores. Para implementar y ejecutar pruebas unitarias existen varias plataformas distintas como NUnit, MSTest o xUnit, pero finalmente se optó por xUnit debido a su sencillez a la hora de declarar nuevas pruebas como se comentó en el apartado 4.1.4. Cabe destacar que las pruebas unitarias se declararán siguiendo una nomenclatura del estilo Método_Estado_ComportamientoEsperado. Por ejemplo, un test de una función en la que se comprueba un valor se llamaría ComprobarValor_ValorErróneo_GeneraExcepción.

Configurar el entorno de CI	
Descripción	*COMO* usuario o product owner *QUIERO* disponer en cualquier momento de la última versión estable del proyecto *ENTONCES* necesito que por cada commit en master se compile automáticamente toda la solución y reporte el resultado.
Tareas	1. Investigación previa. 2. Implementar los cambios necesarios en la solución de VS. 3. Implementar el proceso de CI (Contiuous Integration) en GitLab. 4. Documentación.
Criterios de aceptación	1. El proceso debe lanzarse por cada commit/merge en master. 2. El proceso debe de reportar de alguna forma si la compilación ha terminado de forma correcta o no. 3. Si todo ha compilado correctamente deben de lanzarse automáticamente los unit tests. 4. Si los unit tests fallan el proceso de CI debe darse for fallido.

Tabla 5.6: Historia de usuario n.º 7

Los procesos de CI/CD vistos en el apartado 3.2 se realizan a través de un sistema lineal conocido como *pipeline*, donde se pasa por cada fase de forma ordenada comprobando que el resultado de cada una es correcto antes de continuar con la siguiente. GitLab incluye su propio sistema configurable para cada repositorio, donde con cada modificación en el repositorio se realiza automáticamente la compilación y las pruebas unitarias sobre el nuevo código, mostrando su resultado para poder corregir los posibles errores. Además, también ofrece la posibilidad de disponer de la versión del software generada al final del proceso, facilitando el proceso de despliegue del producto.

Para poder utilizar el sistema CI/CD de GitLab lo primero es incluir un fichero llamado ".gitlab-ci.yml" con la configuración deseada. Este fichero con formato *YAML* contiene las definiciones de las distintas fases y trabajos a ejecutar de forma ordenada en el *pipeline*, y está configurado para que se ejecuten cuando se realice un *commit* o *merge* sobre la rama *master* del repositorio. Las tres fases definidas son:

- ***Build***: para esta fase se define un único trabajo con el mismo nombre en el que se compilará la solución mediante la herramienta de línea de comandos dotnet. Si la compilación finaliza sin ningún error se pasará a las pruebas unitarias.
- ***Test***: contará con una única tarea en la que se ejecutarán las pruebas unitarias definidas en el proyecto de xUnit.net de la solución. Si todas las pruebas son superadas el trabajo finalizará, y se completará con ello el proceso de integración continua de la aplicación.
- ***Publish***: se publicará la aplicación UIComposer indicando su versión. Para ello se utilizará GitVersion y el comando "dotnet publish", donde se indicará mediante parámetros la configuración *Release* para compilar, que la aplicación será autocontenida (se incluirán las dependencias .NET), y que el sistema operativo en el que se usará será Windows de 64 bits. Al finalizar esta fase se culminará la entrega continua.

Una vez se han configurado los procesos es necesario tener un sistema donde ejecutarlos. Dentro de GitLab existen varias opciones distintas, y en este caso se ha elegido utilizar un *Runner*. Una vez éste haya sido instalado como un servicio en el entorno de desarrollo Windows y registrado en el repositorio, cuando se realice una nueva modificación se revisará el fichero de configuración, y, en caso de que su especificación lo indique, el *Runner* ejecutará los trabajos definidos en él.

Visualizar un JPG en la aplicación	
Descripción	*COMO* usuario *QUIERO* poder visualizar imágenes JPG en la aplicación *ENTONCES* necesito implementar la carga y visualización de imágenes JPG en el proyecto.
Tareas	1. Implementar un visualizador de JPG dentro de la aplicación. 2. Abrir ficheros JPG especificos. 3. Mostrar el JPG seleccionado dentro del visualizador. 4. Unit testing. 5. Documentación. 6. Pull request.
Criterios de aceptación	1. El contenido de la imagen debe poder visualizarse por completo en el contenedor. 2. Se debe de poder escoger el fichero JPG de forma visual. 3. Si se abre un segundo fichero JPG este debe sustituir al primero.

Tabla 5.7: Historia de usuario n.º 8

El propósito de esta tarea es que la aplicación cuente con la funcionalidad necesaria para poder mostrar una imagen. Aunque este no sea el objetivo de la herramienta, sirve como prueba de concepto para el visualizador donde se deberá presentar la interfaz de usuario que se esté editando. Con esta tarea también se diseñó la ventana principal siguiendo el diseño establecido en el Mockup.

En la reunión de final de Sprint se comprobó que se finalizaron todas las tareas excepto la historia de usuario número 1 que se pasó al siguiente. También se decidió cambiar de Visual Studio 2017 a 2019, debido a que esta nueva versión incluye un mejor soporte para la plataforma .NET Core 3.

5.2 Sprint 2

Los objetivos de este Sprint son completar la configuración del *pipeline* para alcanzar el despliegue continuo, establecer la arquitectura extensible de la aplicación e implementar el primer plugin de HTML. El Sprint comenzó el 11/02/2019 y finalizó el 01/03/2019, las historias de usuario realizadas fueron las siguientes:

Investigar control WPF para visualizar HTML	
Descripción	*COMO* desarrollador *QUIERO* poder visualizar contenido HTML en el visor de la aplicación *ENTONCES* necesito realizar un estudio previo para verificar si algún control de WPF permite renderizar directamente código HTML.
Tareas	1. Investigación inicial controles renderizar HTML. 2. Documentar los resultados en la Wiki. 3. Implementar spike que permita renderizar un HTML.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. El spike debe permitir renderizar un HTML y visualizarlo. 3. El spike debe de estar en la carpeta spikes del repositorio, en master.

Tabla 5.8: Historia de usuario n.º 1

Con esta tarea se pudo experimentar con algunas opciones para renderizar contenido HTML con un control WPF. Se exploró la posibilidad de usar el control WebView, un control moderno que utiliza el motor del navegador Microsoft Edge, pero este se encuentra sólo disponible bajo UWP y no WPF. Finalmente se comprobó que la opción más estable era el control WebBrowser, el cual es en realidad sólo un envoltorio alrededor de una versión ActiveX de Internet Explorer.

El mayor problema que presenta este control es que tampoco es en realidad un control nativo de WPF, si no que se trata de un control de Win32. Al ser tecnologías diferentes cada una se ocupará de dibujar la región que ocupen sus controles en la ventana, que se conoce como *airspace*. En este caso las guías que se utilizan para redimensionar las columnas se dibujan con un color semitransparente por encima de la columna central, y en esta columna se encuentra el control WebBrowser, con lo que se produce una violación del espacio reservado representado en la figura 5.2.

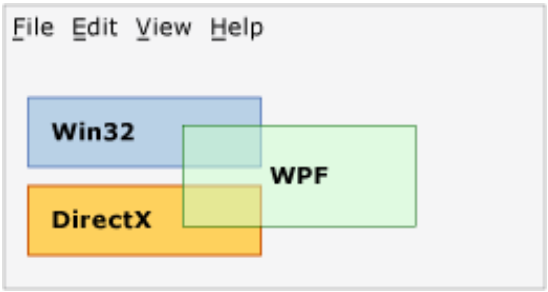

Fig. 5.2. Violación espacio reservado.

Definir la arquitectura de gestión de visualizadores	
Descripción	*COMO* cliente *QUIERO* poder visualizar el contenido de diferentes tipos de formatos de definición de UI en la aplicación *ENTONCES* necesitamos definir un modelo de aplicación que permita manejar un número indeterminado de visualizadores.
Tareas	1. Investigación inicial. 2. Definir API de los visualizadores. 3. Definir cómo se cargarán los visualizadores en la aplicación (como Assemblies). 4. Definir cómo se cargarán los controles de los visualizadores en la sección central de la UI. 5. Documentación. 6. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Deben generarse diagramas de UML - secuencia para la las tareas de carga de los assemblies y controles. 3. Debe generarse diagrama de clases de arquitectura general. 4. La solución debe de permitir cargar un número indeterminado de visualizadores (todos los que se implementen). 5. El API definido debe de ser consistente para todos los visualizadores. 6. Diagrama UML de clases para los visualizadores.

Tabla 5.9: Historia de usuario n.º 9

Con esta tarea se pretende cumplir con uno de los objetivos establecidos para la herramienta UIComposer, la extensibilidad. Esta se conseguirá mediante el uso de complementos o *plugins* que añadirán la capacidad para visualizar otros tipos de formatos de definición de interfaces. Para ello se utilizará la biblioteca *Managed Extensibility Framework* (MEF), la cual permite detectar nuevos componentes para la aplicación sin tener que aplicar ninguna configuración previa.

Un componente MEF especifica en su declaración tanto sus dependencias (importaciones) como qué funcionalidades (exportaciones) están disponibles. Cuando se crea un elemento, el motor de composición de MEF cubre sus importaciones, con lo que está disponible en otros elementos. MEF permite a las aplicaciones detectar y examinar elementos por sus metadatos, sin crear instancias ni cargar sus ensamblados. Por consiguiente, no hay necesidad de especificar meticulosamente cuándo y cómo deben cargarse las extensiones.

A parte de MEF el otro elemento más importante serán las APIs (*Application Programming Interface*) definidas para los plugins: IPlugin e IPluginMetadata. La primera incluirá las funciones a implementar por los plugins, y la segunda la información que permitirá identificarlos para ser importado con MEF desde la aplicación. Ambas se definen en el proyecto Lib de la solución, dentro del fichero PluginAPI.cs. La relación de un plugin con las interfaces se puede observar en la figura 5.3.

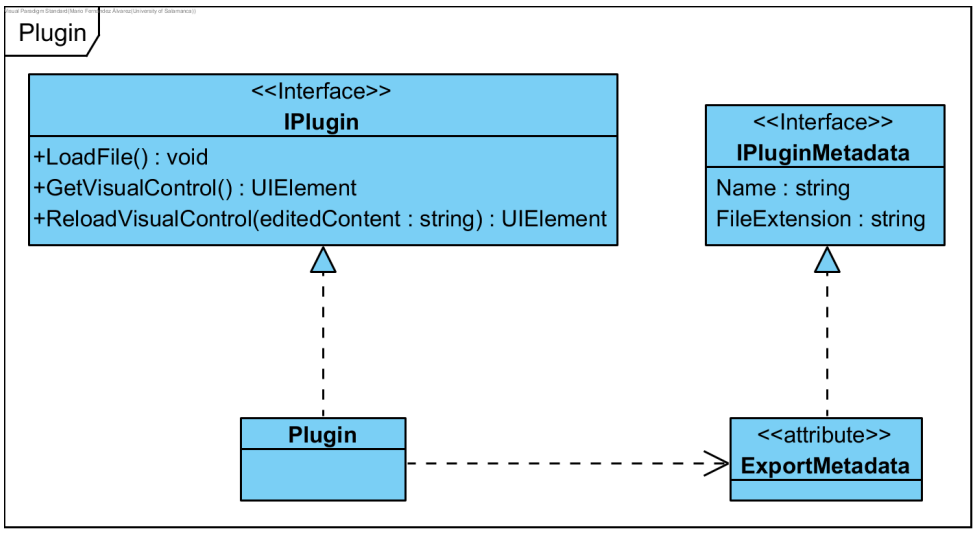

Fig. 5.3. Plugin en la versión 0.2.0

La ventana principal será el componente central de la aplicación, ya que esta será la encargada de cargar los ficheros y de realizar las peticiones a los plugins para obtener sus visualizadores. La relación entre los distintos módulos de la aplicación se ha representado con un diagrama UML de componentes en la figura 5.4.

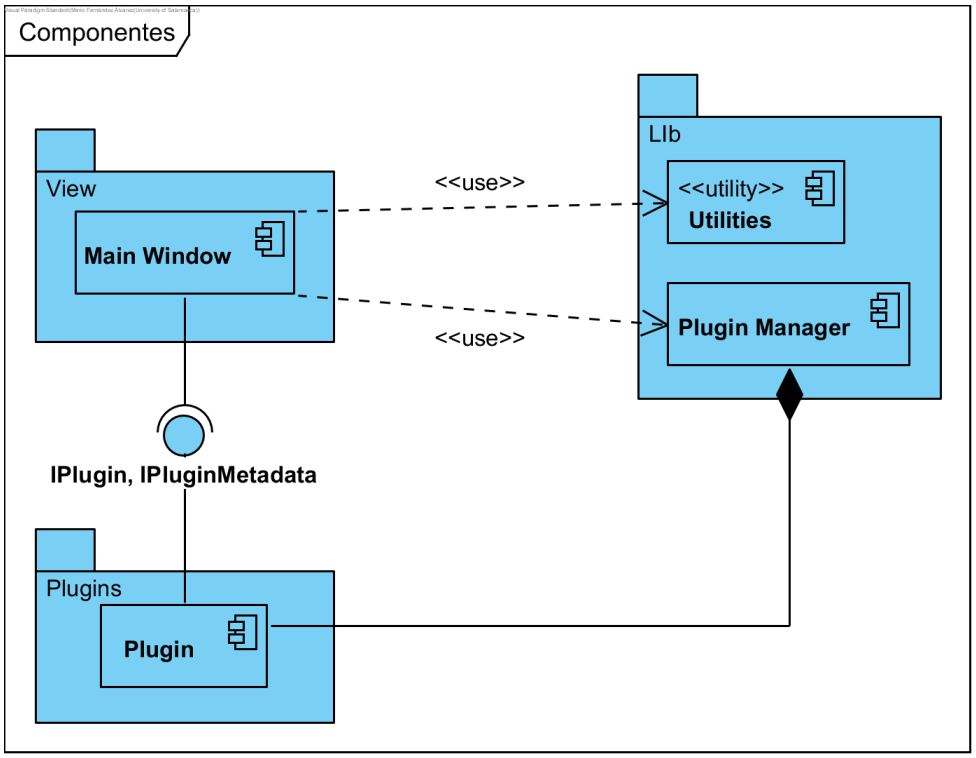

Fig. 5.4. Componentes de la aplicación en la versión 0.2.0

El proceso de carga de plugins será gestionado por un objeto de la clase “Plugin Manager”, el cual se encargará de importar los plugins desde el directorio “Plugins” que se encontrará en la misma ubicación que la aplicación UIComposer. Este proceso sigue los pasos mostrados en el diagrama de secuencia de la figura 5.5.

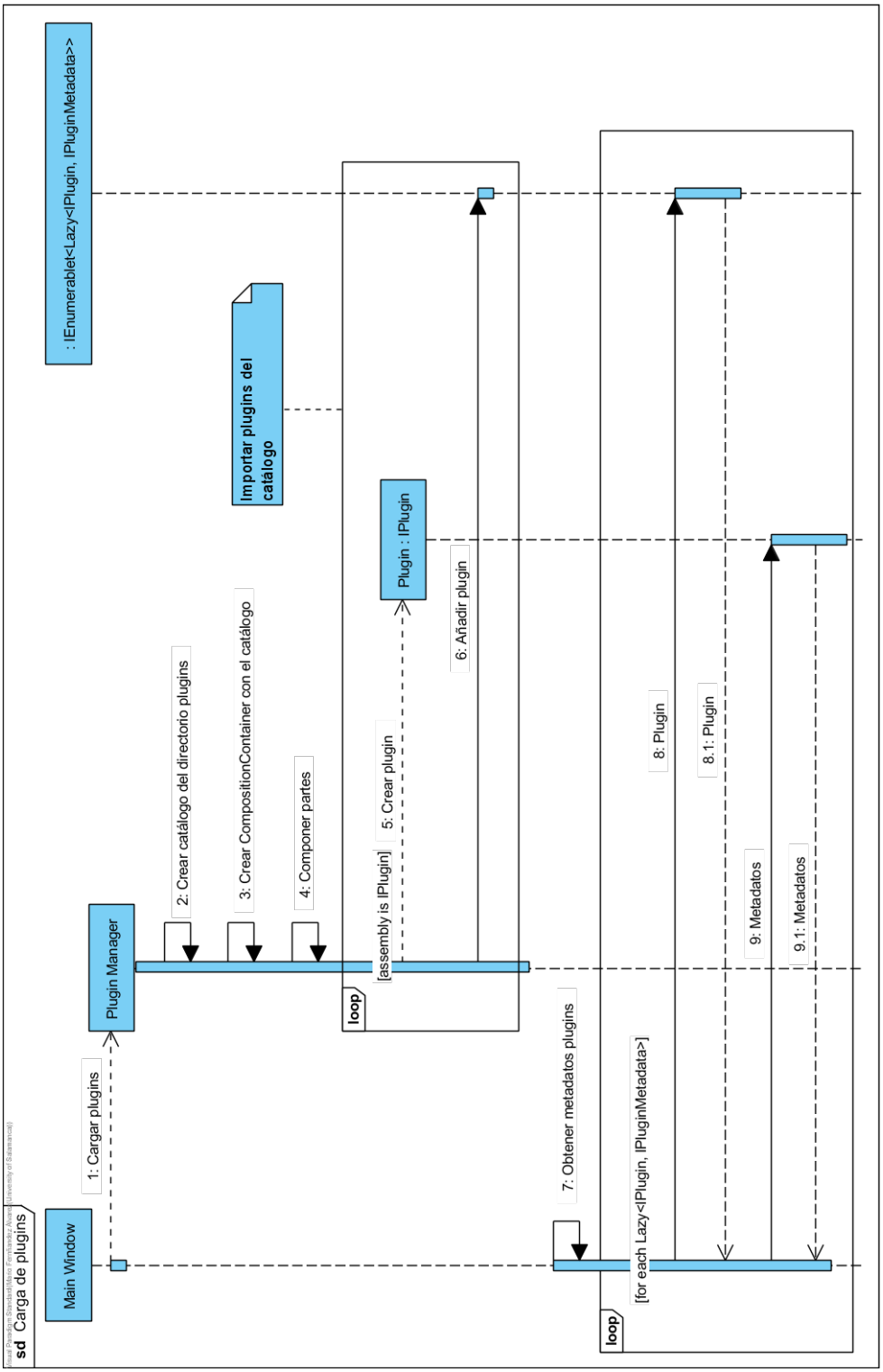

Fig. 5.5. Carga de plugins en la versión 0.2.0

Tras haber cargado todos los plugins presentes en el directorio "Plugins", la aplicación podrá indicar gracias a sus metadatos todos los formatos disponibles para usar. Luego, una vez el usuario elija el fichero a abrir, la aplicación deberá solicitar al plugin apropiado el control visual a mostrar siguiendo el proceso descrito en la figura 5.6.

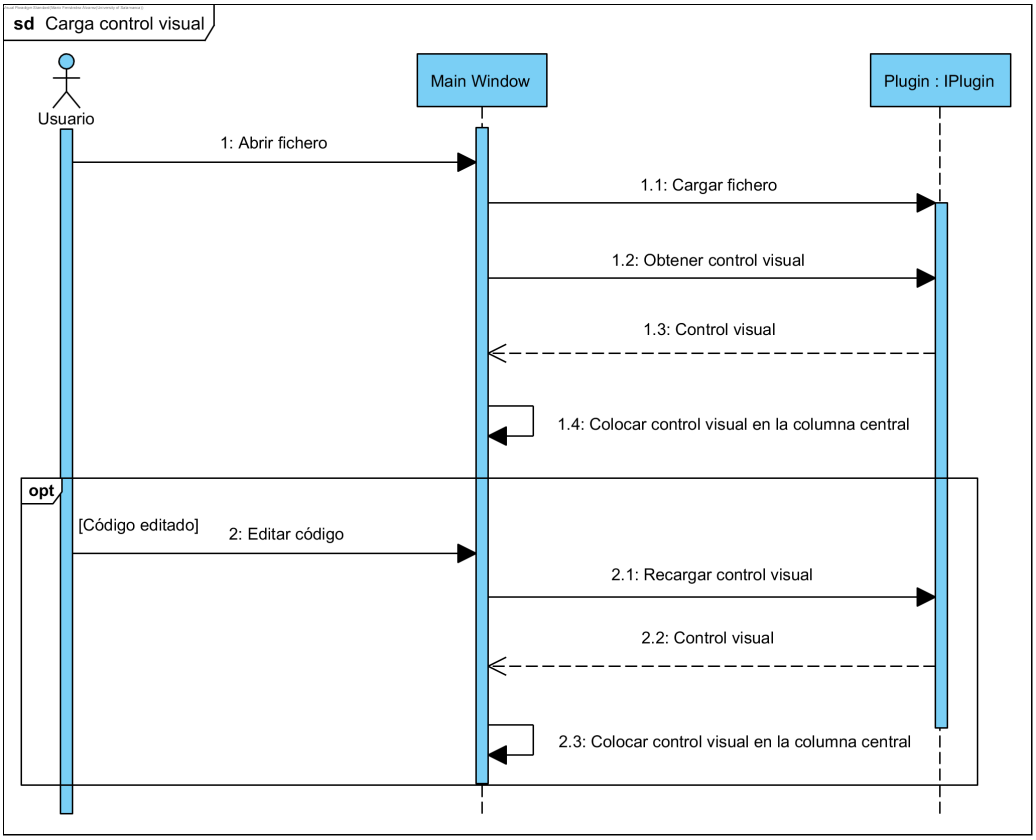

Fig. 5.6. Carga control visual en la versión 0.2.0

Publicar release fin de sprint	
Descripción	*COMO* usuario *QUIERO* disponer de la última versión oficial del programa en cualquier momento *ENTONCES* la release oficial de cada sprint debe de publicarse y estar accesible desde la Web del proyecto.
Tareas	1. Investigación inicial. 2. Definir un lugar donde alojar las releases. 3. Implementar un proceso de CD usando GitLab. 4. Verificación. 5. Documentación.
Criterios de aceptación	La release debe de estar disponible desde la página de GitLab.

Tabla 5.10: Historia de usuario n.º 10

La configuración establecida en el anterior Sprint no cumple con el requisito principal de despliegue continuo: hacer que la aplicación actualizada esté disponible para su descarga automáticamente. Para conseguirlo se añadió un nuevo trabajo en la fase *publish* en el que se ejecuta un script de Powershell. En este script se actualizan las placas de la página del repositorio para que muestren el estado del *pipeline* y la última versión disponible para descargar, como se puede ver en la figura 5.7.

Fig. 5.7. Placas del repositorio

Implementar visualizador HTML	
Descripción	*COMO* cliente *QUIERO* visualizar el renderizado de un HTML en la aplicación *ENTONCES* necesito implementar un control de visualización que permita el manejo de HTMLs.
Tareas	1. Crear el proyecto base en VS. 2. Crear clase 'HtmlVisualizador', que implementa 'IVisualizador'. 3. Implementar los diferentes métodos de carga de HTML. 4. Implementar los diferentes métodos de renderizado de HTML. 5. Implementar los diferentes métodos de obtención del código asociado. 6. Añadir el visualizar a la aplicación. 7. Unit testing 8. Documentación. 9. Pull request.
Criterios de aceptación	1. La implementación debe seguir lo definido en los UML de la historia de usuario n.º 9. 2. El renderizado debe de mostrarse en el panel central de la aplicación. 3. El visualizador debe de soportar formato HTML. 4. El código HTML (cómo texto) debe de mostrarse en el panel izquierdo de la aplicación. 5. Toda la documentación debe de estar disponible desde la Wiki.

Tabla 5.11: Historia de usuario n.º 11

Gracias a la investigación previa ya se puede proceder a implementar un nuevo plugin que permita visualizar ficheros con contenido HTML. El nuevo proyecto de tipo WPF añadido a la solución se llamará "HTMLPlugin" y contendrá un único fichero en el que se definirá la clase del mismo nombre. Esta clase implementará las interfaces de la API de plugins, IPlugin e IPluginMetadata, para que pueda ser importado por la aplicación, y utilizará el control WebBrowser para mostrar el fichero HTML cargado. Con esta tarea se aumentó la versión a la 0.2.1.

También es necesario incluir los plugins en el proceso de CD para que estén disponibles junto con la aplicación. Para ello se ejecutará durante la fase *publish* un nuevo script de PowerShell que buscará todos los directorios que concuerden con el patrón "*Plugin" y ejecutará la orden "dotnet publish" en cada uno de ellos. Con esto se generará un fichero .dll por cada proyecto de plugin en la solución, y se copiará al directorio "Plugins" junto con la aplicación para que puedan ser descargados todos juntos.

Incluir plugins en release	
Descripción	*COMO* usuario *QUIERO* tener los plugins junto con la aplicación principal *ENTONCES* se deben incluir los plugins en el proceso de CI/CD.
Tareas	1. Realizar los cambios necesarios en los proyectos de plugins. 2. Modificar los trabajos del proceso de CI/CD. 3. Documentación. 4. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Los plugins deben estar incluidos en la release final.

Tabla 5.12: Historia de usuario n.º 13

Al final del Sprint se realizó la revisión para valorar el trabajo realizado. Se completaron las tareas de implementación del plugin para HTML y se actualizó el sistema de CI/CD, aunque se tuvo que aplazar la investigación de la visualización de XAML para el siguiente Sprint.

5.3 Sprint 3

El objetivo principal de este Sprint será el de desarrollar un plugin para el formato XAML que permite visualizar ficheros con los elementos básicos. También se tratará de resolver algunos errores menores de la interfaz en la ventana principal. El Sprint comenzó el 04/03/2019 y finalizó el 25/03/2019. Las historias de usuario realizadas fueron las siguientes:

La primera tarea llamada “Investigar control WPF para visualizar XAML” consiste en una investigación previa sobre como visualizar el contenido de un fichero XAML. Dado que la aplicación se desarrolla también usando XAML la propia plataforma de .NET Core 3 contiene las herramientas necesarias para llevarlo a cabo. Para conseguirlo se implementó un *spike* de prueba donde se demuestra el uso de la clase “XAMLReader” para leer el contenido de un fichero y generar un árbol visual con todos los elementos.

El principal problema que se presenta es que los ficheros XAML se asocian a un fichero de C# donde se define la lógica, entonces cuando se intenta leer un fichero y no se encuentra los métodos asociados en esa otra clase se produce un error. Para evitar que esto suceda se debe implementar una clase que herede de “XAMLReader” en la que se omitan los datos del fichero asociados a otra clase. Con esto se consigue un “Loose XAML”, el cual se caracteriza por:

- El elemento raíz no es una ventana. Puede ser cualquier otro tipo como un control de usuario o un panel.
- El elemento raíz debe contener todos los espacios de nombres necesarios.
- El elemento raíz no puede tener un atributo de clase con el que se asocie a otro fichero de C#.
- Ningún elemento puede tener controladores de eventos.

Añadir scrollbar a la sección de código	
Descripción	*COMO* usuario *QUIERO* desplazar el contenido del fichero con código verticalmente *ENTONCES* se debe añadir un scrollbar en la columna izquierda.
Tareas	1. Añadir scrollbar en la columna izquierda. 2. Documentación. 3. Merge request.
Criterios de aceptación	1. El scrollbar debe aparecer tras cargar el fichero. 2. El scrollbar debe permitir desplazar verticalmente el texto. 3. Los resultados deben de estar en la Wiki.

Tabla 5.13: Historia de usuario n.º 14

En el Sprint anterior se añadió a la columna izquierda de la ventana principal la sección donde mostrar el contenido del fichero abierto. Esto se consigue mediante un control *TextBox* de WPF que permite seleccionar y editar el texto que contenga. El problema es que este control no muestra una barra para poder deslizar el texto verticalmente, así que se modificó para habilitar la barra una vez se muestre el contenido del fichero.

Implementar visualizador XAML básico	
Descripción	*COMO* cliente *QUIERO* visualizar ficheros XAML sencillos en la aplicación *ENTONCES* necesito implementar un Plugin que permita visualizar ficheros XAML básicos.
Tareas	1. Crear el proyecto en la solución de VS. 2. Crear la clase XAMLPlugin con las interfaces IPlugin e IPluginMetadata. 3. Implementar los métodos para cargar un fichero XAML. 4. Implementar los métodos para crear un visualizador de XAML. 5. Unit testing. 6. Documentación. 7. Merge request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. El visualizador debe mostrarse en la columna central. 3. El visualizador debe soportar ficheros XAML básicos.

Tabla 5.14: Historia de usuario n.º 15

El primer paso es crear un nuevo proyecto en la solución llamado “XAMLPlugin” de tipo WPF, que contará con un único fichero “XAMLPlugin.cs” en el que se definirá la clase homónima que implementará las interfaces de la API de plugins. Para cargar un fichero XAML y poder generar el visualizador se seguirá el procedimiento visto en el *spike* anterior, aunque habrá que realizar un cambio. Cuando se trate de un fichero en el que se defina una ventana se deberá obtener el contenido de esa ventana para mostrarlo en el visualizador, ya que en WPF no es posible mostrar una ventana dentro de otra.

Con esta implementación se consigue obtener el visualizador correctamente, pero surge un problema a la hora de cerrar la aplicación. Esto se debe a que el motor de renderizado XAML crea una serie de hilos al procesar el fichero, y estos hilos impiden el cierre ordenado de la aplicación. Este fallo deberá ser revisado durante la reunión de final de Sprint.

Modificar los GridSplitters para que permitan previsualizar el tamaño de las columnas	
Descripción	*COMO* usuario *QUIERO* previsualizar el tamaño de las columnas de la aplicación al cambiarlo *ENTONCES* necesito modificar los GridSplitters para que permitan la previsualización.
Tareas	1. Modificar el estilo de los GridSplitters. 2. Modificar el estilo de previsualización de los GridSplitters. 3. Fijar la posición de los GridSplitters para que siempre sean visibles. 4. Unit testing. 5. Documentación. 6. Merge request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. La previsualización debe verse claramente. 3. La previsualización debe mostrarse sobre el resto de contoles visuales.

Tabla 5.15: Historia de usuario n.º 18

Con esta tarea se pretende solventar el problema comentado en la historia de usuario n.º 1 sobre el dibujado de los controles visuales en regiones con tecnologías distintas. La intención inicial era la de modificar el estilo y comportamiento de las guías de redidmensionado para que no sobrepasasen la columna central donde se coloca el visualizador. Sin embargo esto no era posible ya que causaba graves problemas de rendimiento en la aplicación, debido a que esto obliga a redibujar las columnas cada vez que se desplazaran las guías.

Al no poderse completar esta tarea el problema con el renderizado de las guías persiste. Por ello se añadió una tarea al *backlog* para explorar de nuevo otras opciones para el visualizado de HTML que no presenten este error.

Durante la reunión de final Sprint se evaluó positivamente el trabajo realizado ya que se puedieron completar casi todas las tareas. Durante la demostración se pudo comprobar el fallo en el cierre de la aplicación tras cargar un fichero XAML, y se añadió una nueva tarea para arreglarlo. También se observó que el visualizador XAML permite la interacción con sus controles a menos que se deshabilite completamente. Esto afecta a la presentación de los controles, por lo que se añadió otra tarea para plantear alguna otra opción.

5.4 Sprint 4

Durante este periodo se tratará de solventar los errores en el plugin XAML vistos al final del Sprint anterior, y también en el plugin HTML por el uso del control WebBrowser. Además se actualizarán los diagramas UML y la documentación del proyecto de acuerdo a los últimos cambios realizados. El Sprint comenzó el 25/03/2019 y finalizó el 15/04/2019. Las historias de usuario realizadas fueron las siguientes:

Investigar creación/cierre threads UserRequest con el visualizador de XAML	
Descripción	*COMO* desarrollador *QUIERO* que la aplicación se cierre correctamente tras abrir un fichero XAML *ENTONCES* tengo que gestionar los hilos creados durante la carga del XAML.
Tareas	1. Identificar los hilos creados durante la carga del fichero. 2. Cerrar los hilos que permanezcan bloqueados por ÜserRequest". 3. Fijar la posición de los GridSplitters para que siempre sean visibles. 4. Documentación. 5. Merge request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. No deben quedar hilos bloqueados. 3. La aplicación debe cerrarse correctamente.

Tabla 5.16: Historia de usuario n.º 20

El origen de los hilos resultó ser el objeto *Dispatcher* asociado a cada control visual creado en WPF. Este objeto es el encargado de gestionar la cola de eventos generados por la interacción de usuario con la interfaz, y al cargar en tiempo de ejecución un nuevo fichero XAML este genera su propio *Dispatcher*. Como este no está asociado con el de la aplicación UIComposer, cuando se intenta cerrar esta los hilos que se encuentran en la cola correspondiente al XAML cargado no se cierran y paralizan la aplicación.

Para solucionar esto se añadió la interfaz *IDisposable* a la api de plugins, de modo que cada plugin debe implementar los métodos necesarios para poder desechar correctamente todos sus recursos. Sigiuendo este proceso, cuando se cierre la ventana principal de la aplicación UIComposer esta solicitará al *Plugin Manager* que deseche el plugin cargado actualmente para poder gestionar de forma apropiada los recursos que estos estén usando.

En el caso de XAMLPlugin esto supone invocar el cierre del *Dispatcher* cuando el gestor de plugins lo solicite, y con ello se cancelarán todas las tareas pendientes en la cola del *Dispatcher*, es decir, los hilos con los que se gestiona la interacción de la interfaz.

Deshabilitar funcionalidad XAML sin IsEnabled=false	
Descripción	*COMO* usuario *QUIERO* poder visualizar el estilo de los controles XAML *ENTONCES* tengo que deshabilitar la funcionalidad de los controles XAML sin cambiar su estilo.
Tareas	1. Cargar los controles XAML con su estilo. 2. Deshabilitar individualmente cada control XAML. 3. Documentación. 4. Merge request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. El estilo de los controles XAML debe de ser el que se define en el fichero. 3. El usuario no debe poder interactuar con los controles XAML del visualizador.

Tabla 5.17: Historia de usuario n.º 21

La forma más sencilla de deshabilitar la interacción con un elemento de la interfaz es poner a falso la propiedad *IsEnabled*, pero haciendo esto los controles pasan a tener la apariencia de su estilo deshabilitado. Otra opción para evitar esto es cambiar también a falso la propiedad *IsHitTestVisible*, con lo cual en el control no se detectará la interacción con el ratón. Sin embargo esto no es suficiente ya que aún se podría usar el teclado para interactuar con el control, por ejemplo usando el tabulador para seleccionar los controles de la interfaz. Para prevenir esto se debe crear un nuevo método con el que subscribirse al evento *PreviewGotKeyboardFocus*, y en el cuerpo del método indicar que el evento ha sido gestionado para evitar que se propague.

Modificar referencia al plugin usado	
Descripción	*COMO* desarrollador *QUIERO* que la vista principal no referencie al plugin usado *ENTONCES* hay que pasar la referencia del plugin usado de la vista al plugin manager.
Tareas	1. Eliminar referencia en la vista principal. 2. Añadir referencia al plugin en el plugin manager. 3. Añadir los métodos necesarios para poder usar el plugin. 4. Documentación. 5. Merge request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. La vista principal no debe contener una referencia al plugin usado. 3. El plugin manager debe de ser el encargado del plugin usado.

Tabla 5.18: Historia de usuario n.º 25

Hasta ahora la ventana principal contenía una referencia al plugin usado para cargar el fichero, lo cual genera un acoplamiento innecesario entre la vista y el plugin dado que

el *Plugin Manager* es el encargado de gestionar los plugins. En esta tarea se elimina esta referencia y se añaden los métodos necesarios al *Plugin Manager* para que la vista pueda comunicarse con el plugin usado a través de él.

Diseñar diagrama UML de clases arquitectura general	
Descripción	*COMO* desarrollador *QUIERO* tener documentación sobre la arquitectura general de la aplicación *ENTONCES* debo crear un diagrama UML de clases que representa la arquitectura.
Tareas	1. Definir la arquitectura de la aplicación. 2. Crear un diagrama UML de clases que especifique la arquitectura de la aplicación. 3. Documentación. 4. Merge request.
Criterios de aceptación	Los resultados deben de estar en la Wiki

Tabla 5.19: Historia de usuario n.º 22

Con la historia de usuario anterior se eliminó la relación existente entre la vista principal y el plugin usado al cargar un nuevo fichero. Esto puede verse comparando el nuevo diagrama de componentes de la figura 5.8 con el anterior de la figura 5.4.

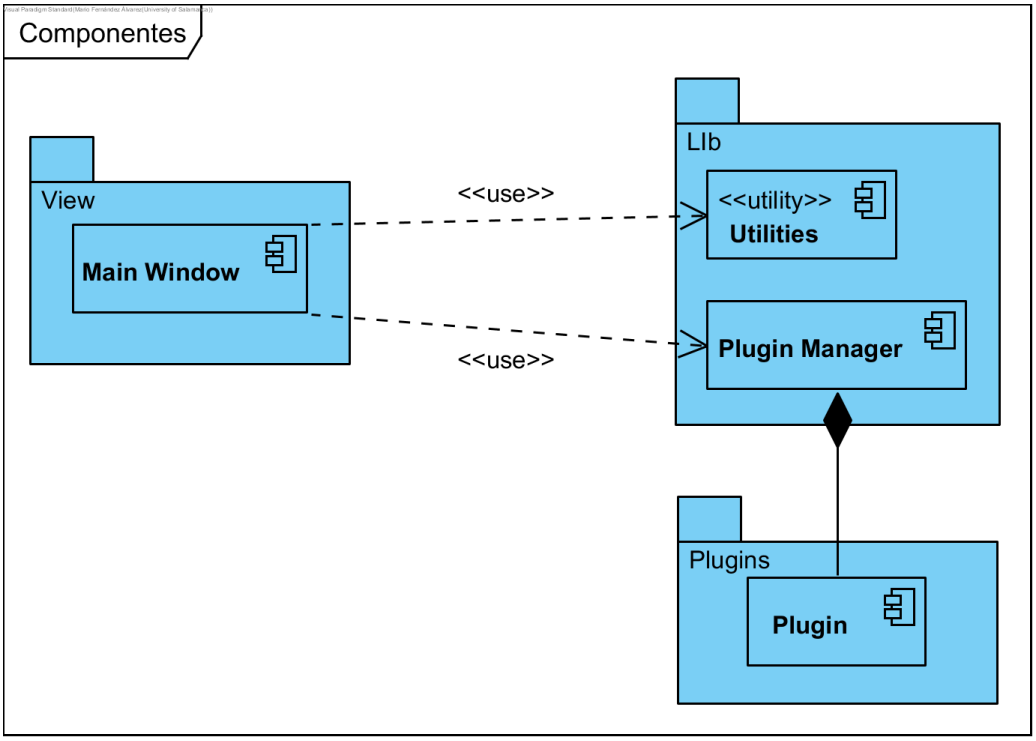

Fig. 5.8. Componentes de la aplicación en la versión 0.4.3

También se creó un diagrama UML de clases en el que se representa la arquitectura completa de la aplicación. En la figura 5.9 se muestran todas las clases que conforman la aplicación incluyendo sus propiedades y sus métodos.

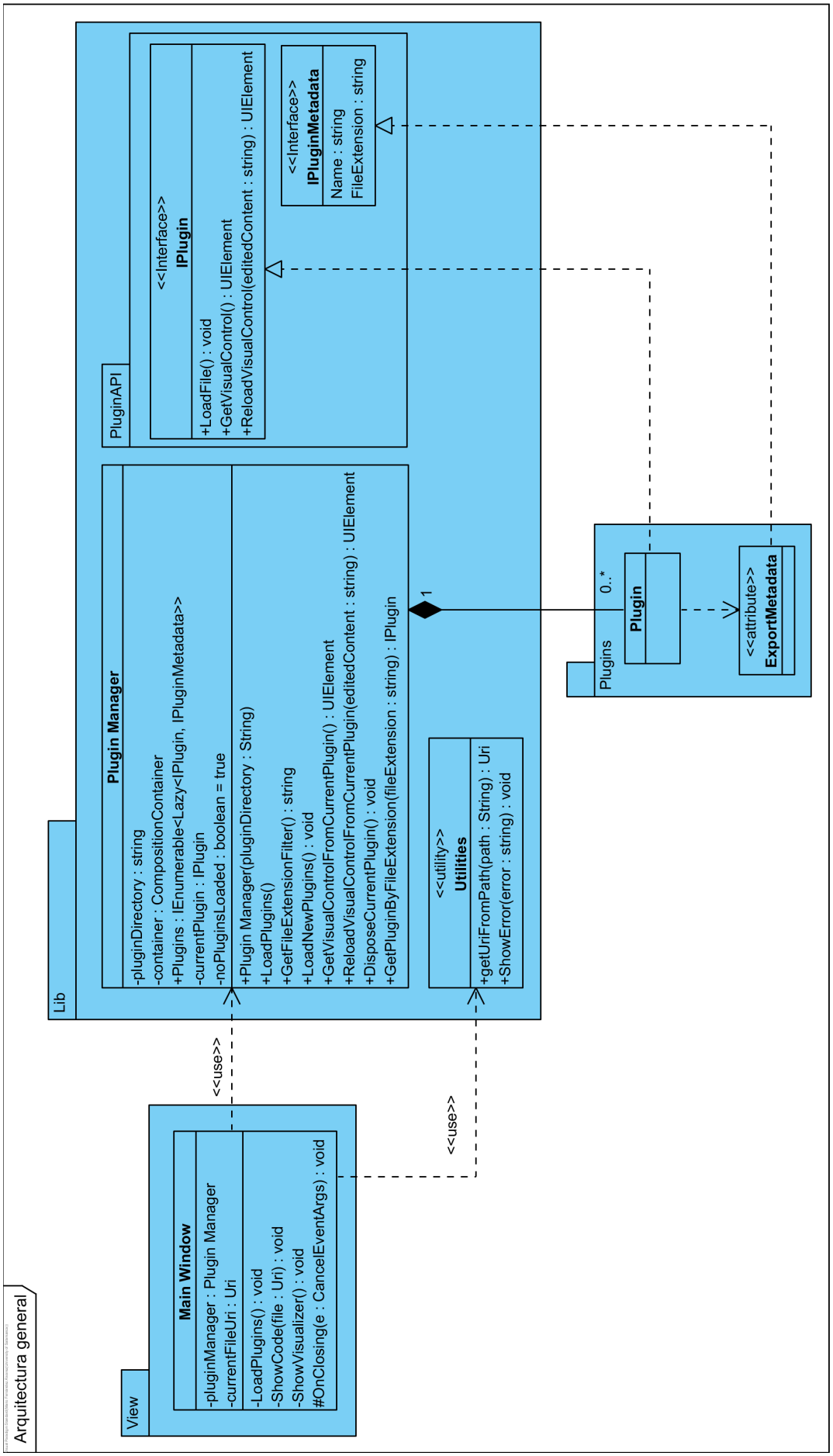

Fig. 5.9. Arquitectura de la aplicación en la versión 0.4.3

Por último se actualizó el diagrama UML de secuencia en el que se muestra el proceso de carga de plugins. En la figura 5.10 se pueden observar los siguientes cambios:

- El *Plugin Manager* debe desechar el plugin cargado anteriormente cuando se proceda a abrir un nuevo fichero.
- La ventana principal solicita el control visual al *Plugin Manager*, el cual se lo retransmite a su vez al plugin cargado.
- La ventana principal deshabilita la funcionalidad del control visual para que el usuario no pueda interactuar con él.

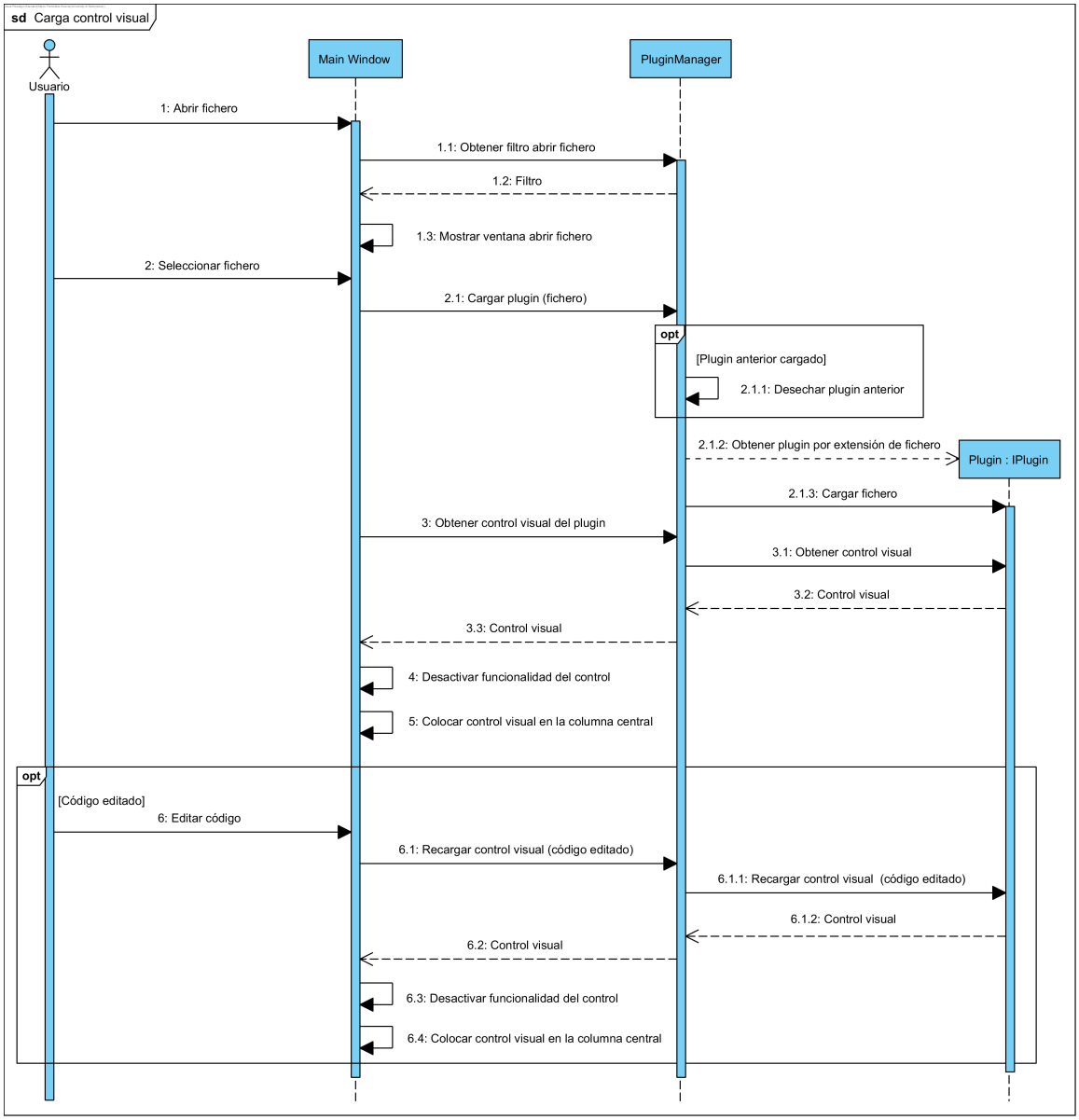

Fig. 5.10. Carga control visual en la versión 0.4.3

Investigar control WebView	
Descripción	*COMO* desarrollador *QUIERO* descubrir el funcionamiento del control WebView *ENTONCES* tengo que implementar un spike para probar el control.
Tareas	1. Investigar el control WebView. 2. Implementar un proyecto spike para comprobar su uso. 3. Documentación. 4. Merge request.
Criterios de aceptación	Los resultados deben de estar en la Wiki

Tabla 5.20: Historia de usuario n.º 24[1]

En esta tarea se plantea el uso del control *WebView* como sustituto del control *WebBrowser* usado en HTMLPlugin para evitar los problemas con las guías de redimensionado. Como ya se ha comentado anteriormente este control utiliza el navegador Microsoft Edge, lo que permitiría también añadir soporte a ficheros con JavaScript moderno. Inicialmente se usó una versión de prueba adaptada para .NET Core 3, y, aunque era funcional se comprobó que no era estable.

La otra alternativa planteada era hacer uso del navegador Chromium desarrollado por Google con la librería *Chromium Embedded Framework*(CEF), la cual permite integrar dicho navegador en otra aplicación. CEF incluye soporte para los lenguajes de programación C y C++, pero hay proyectos externos que la adaptan para otros lenguajes y plataformas. Para .NET Core 3 existe el proyecto *CEFGlue*, el cual se usó en un *spike* para comprobar su funcionamiento.

Con el *spike* se pudo demostrar el uso de CEF para abrir ficheros HTML, pero también presentaba un problema claro, y es que CEF requiere iniciar el motor de Chromium durante el arranque de la aplicación. No es posible iniciar el navegador una vez se haya cargado la interfaz de la aplicación, lo que hace que no sea apto para la aplicación UIComposer ya que en ella la carga de un plugin se debe hacer dinámicamente al abrir un fichero.

En la reunión de final del Sprint se comprobó que los errores en XAMLPlugin fueron resueltos. También se determinó que el control WebBrowser es por el momento la mejor opción para visualizar HTML, si bien presenta algunos problemas. En cuanto a los diagramas UML, se vió que los nuevos cambios en la arquitectura y en el proceso de carga de plugins se reflejaron correctamente.

Además durante este Sprint también se produjeron algunos errores en el proceso de CI/CD, concretamente en la fase *build*. Estos errores provocan que el *pipeline* falle, por lo que deben ser resueltos para poder continuar con el trabajo.

[1]En el repositorio de GitLab la historia de usuario aparece en el Sprint 5, pero en realidad se completó durante el Sprint 4.

5.5 | Sprint 5

En este Sprint será necesario solucionar en primer lugar los problemas con los procesos de CI/CD. Después se completará el trabajo con XAMLPlugin para que permita visualizar archivos XAML más complejos. El Sprint comenzó el 15/04/2019 y finalizó el 06/05/2019. Las historias de usuario realizadas fueron las siguientes:

Resolver error pipeline CI/CD	
Descripción	*COMO* desarrollador *QUIERO* poder usar el sistema CI/CD de forma efectiva y sin errores *ENTONCES* tengo que solucionar los errores que ocurren en la ejecución de los procesos de CI.
Tareas	1. Identificar el error en la fase 'Build'. 2. Resolver el error. 3. Verificar que el funcionamiento es correcto.
Criterios de aceptación	El sistema CI/CD debe ejecutarse sin errores.

Tabla 5.21: Historia de usuario n.º 26

En la fase *build*, tras compilar cada plugin estos se copian automáticamente a los directorios donde se encuentran la aplicación principal y las pruebas unitarias. Esto se permite depurar la aplicación y ejecutar las pruebas unitarias de los plugins directamente. El error en este caso se produce porque el orden de compilación era incorrecto, entonces las órdenes de copia fallan al no poder encontrar los directorios correspondientes. Para solucionarlo se pasaron las órdenes de copia a los proyectos *View* y *UnitTests*.

Tras esto se pudo comprobar que la fase *test* también fallaba debido a que el *Runner* de GitLab trataba de eliminar los ficheros generados al ejecutar las pruebas pero no podía. Esto se debe a que en .NET estos ficheros se eliminan automáticamente, por lo que hubo que incluir un fichero de configuración para revertir este comportamiento.

Finalmente durante la fase *publish* tampoco se completaba el despliegue. Esto se debe a que cuando se inicia el proceso debido a un *merge* el repositorio pasa a un estado conocido como *detached HEAD*. En este estado el puntero del repositorio git apunta a un *commit* y no a una rama, con lo cual GitVersion no puede determinar la versión actual. Para evitar esto se modificó la configuración para que la fase *publish* sólo se ejecute cuando se realice un *commit* en la rama principal.

El otro objetivo del Sprint es mejorar XAMLPlugin para que permita visualizar ficheros con controles más complejos. Las historias de usuario referentes a este objetivo son "Implementar visualizador XAML avanzado" (n.º 17) y "Definir cada una de las tareas para tener un visualizador XAML 100 % operativo" (n.º 23). Estas se definieron de una forma general durante los Sprints anteriores para englobar este objetivo En las dos historias de usuario siguientes se especifican funcionalidades para XAMLPlugin más concretas.

Eliminar eventos de los controles XAML	
Descripción	*COMO* usuario *QUIERO* abrir ficheros XAML complejos *ENTONCES* es necesario eliminar los eventos de los controles XAML.
Tareas	1. Identificar cada control del fichero leído. 2. Eliminar los eventos asignados al control. 3. Unit testing. 4. Documentación. 5. Merge request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. No se deben producir excepciones al leer el fichero debido a los eventos.

Tabla 5.22: Historia de usuario n.º 28

Como ya se comentó anteriormente los ficheros XAML se asocian a una clase definida en un fichero de C# con la extensión .xaml.cs donde se encuentra la lógica de la aplicación. Con la implementación actual de XAMLPlugin se filtran los atributos XAML que pueden causar errores, pero algunos de los eventos no se identifican correctamente con lo cual solo se pueden cargar ficheros XAML básicos. En esta tarea se modificó el proceso de carga de un fichero XAML para poder tener un mayor control sobre la lectura del fichero siguiendo el proceso descrito en Microsoft, 2017b, representado en el diagrama UML de secuencia en la figura 5.11.

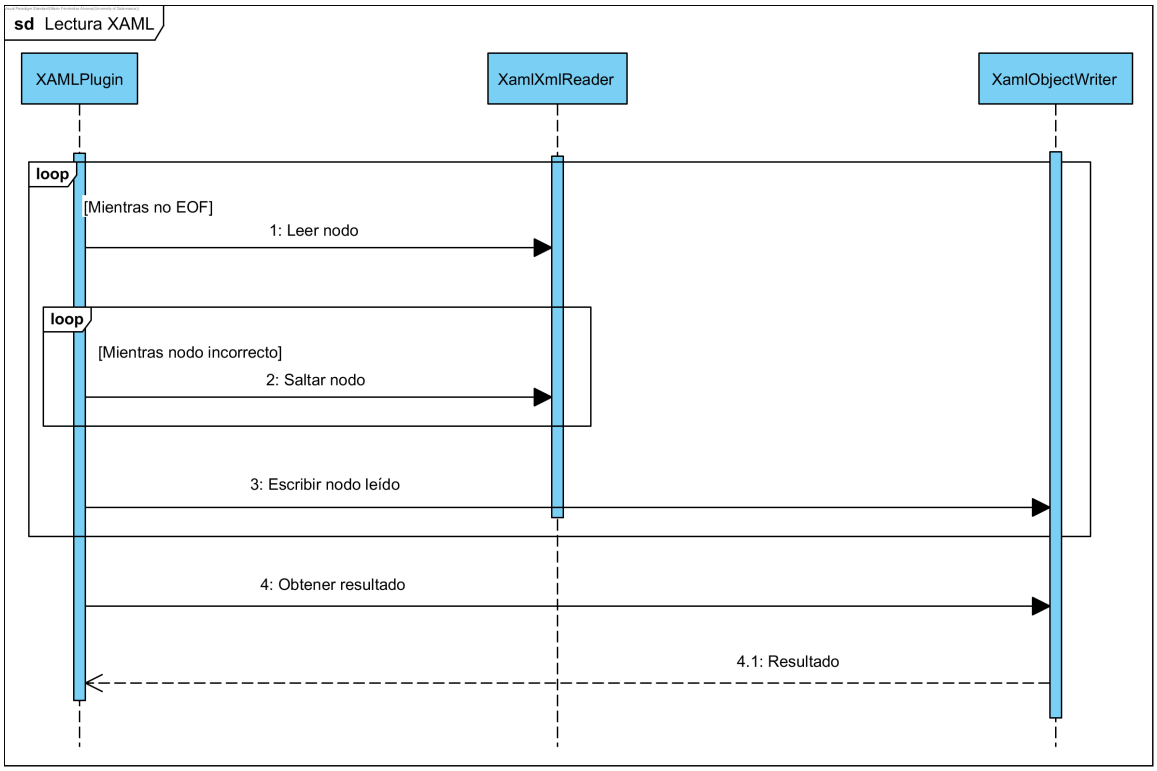

Fig. 5.11. Lectura de XAML en la versión 0.5.1

Cargar ficheros XAML con recursos	
Descripción	*COMO* usuario *QUIERO* cargar otros ficheros XAML externos con recursos asociados al fichero que deseo editar *ENTONCES* tengo que implementar un proceso para cargar otros ficheros XAML con recursos.
Tareas	1. Añadir el nuevo método de carga a PluginAPI. 2. Implementar el método de carga en XAMLPlugin. 3. Unit testing. 4. Documentación. 5. Merge request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. XAMLPlugin debe permitr cargar otros ficheros XAML con recursos. 3. El visualizador de XAMLPlugin debe cambiar en base a los recursos cargados. 4. Los recursos cargados deben poder ser editados también.

Tabla 5.23: Historia de usuario n.º 29

En XAML se pueden definir recursos como cadenas localizables o distintos estilos en un fichero aparte con un diccionario de recursos. Para poder cargar un diccionario de recursos es necesario introducir cambios en la API de plugins que permitan abrir más de un fichero. Para ello se añaden dos posibilidades:

- Proporcionar una lista de ficheros con la extensión y el tipo de cada uno. Así, cuando el usuario decida abrir un fichero se mostrarán tantas ventanas de apertura de fichero como elementos haya en la lista. Por tanto, el proceso de carga de plugins se modifica tal y como se puede ver en la figura 5.12, y también la apertura de ficheros como se puede comprobar en figura 5.13.
- Crear un menú con las acciones que se deseen en cada plugin, el cual será añadido en la barra superior de la ventana principal tras cargar el visualizador.

En XAMLPlugin se ha optado por la segunda opción porque si al cargar un fichero XAML no se encuentra algún recurso sólo se genera una advertencia en el lector, es decir, no impide generar el árbol visual con los controles del fichero XAML. Para cargar el fichero con el diccionario de recursos se añadirá una entrada al menú del plugin, con la cual se abrirá una ventana para seleccionar el fichero con el diccionario. Cuando se cargue el fichero se añadirán sus recursos al visualizador, de modo que cualquier estilo definido en él se aplicará automáticamente.

En la revisión del Sprint se verificó que los fallos del *pipeline* fueron resueltos y que XAMLPlugin permite la apertura de ficheros complejos, además de añadir la nueva funcionalidad de los diccionarios de recursos. Pero durante la demostración surgió un problema al intentar cargar un fichero XAML tras haber cargado ya otro previamente, lo cual se añadió al *backlog* para el siguiente Sprint.

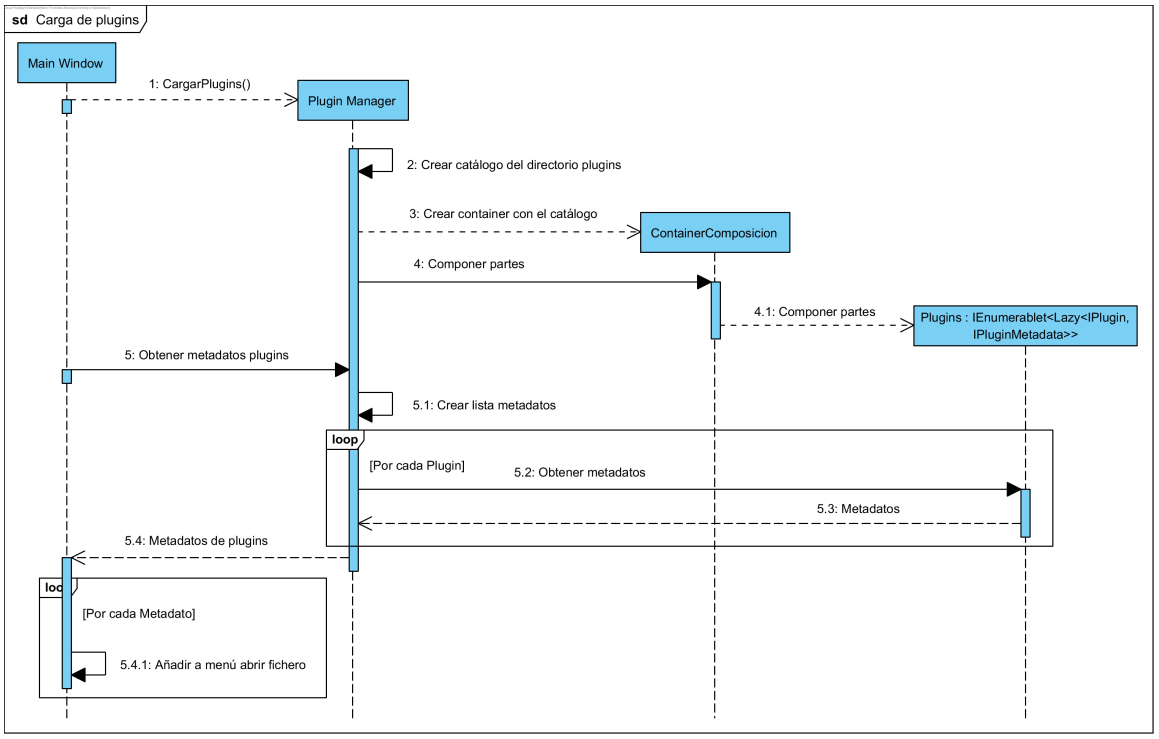

Fig. 5.12. Carga de plugins en la versión 0.5.2

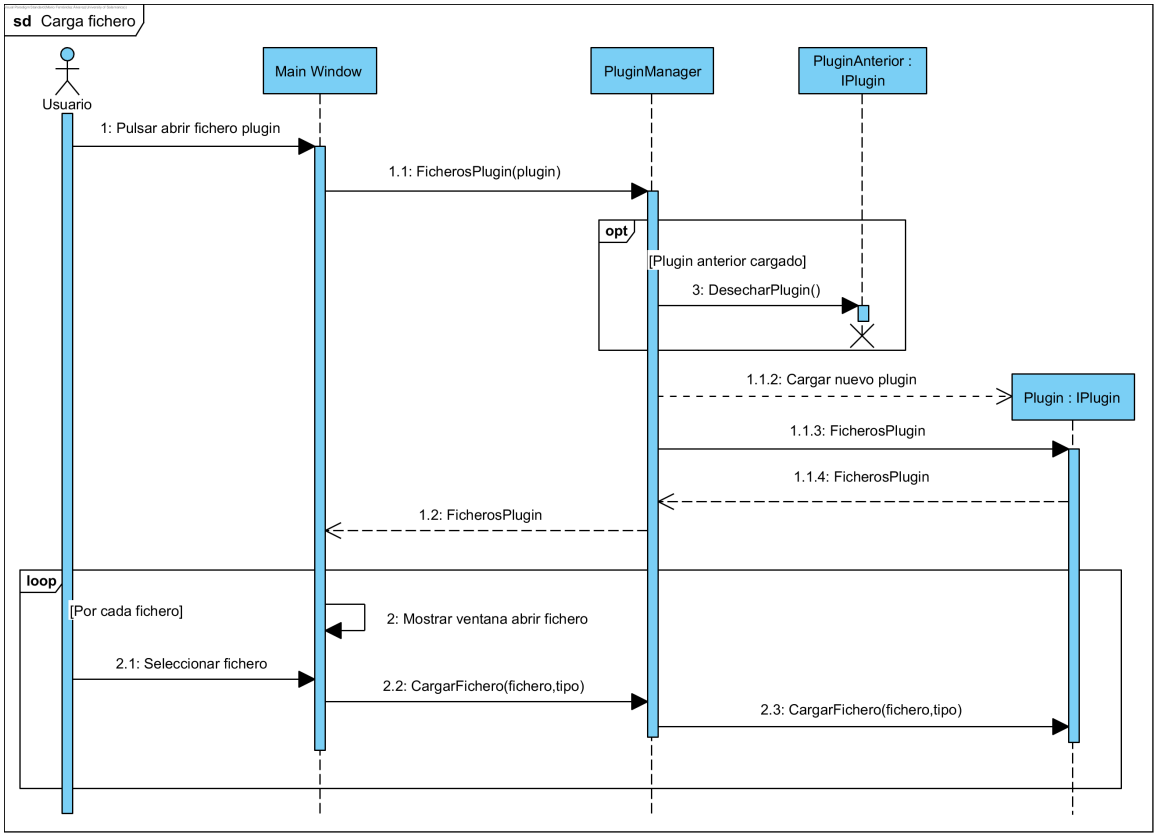

Fig. 5.13. Carga fichero en la versión 0.5.2

5.6 Sprint 6

El objetivo del sexto Sprint será el de implementar el panel de propiedades, que permitirá editar las propiedades de cada elemento definido en el fichero de UI cargado. El Sprint comenzó el 06/05/2019 y finalizó el 27/05/2019. Las historias de usuario realizadas fueron las siguientes:

Implementar panel de propiedades básico	
Descripción	*COMO* usuario *QUIERO* poder ver y editar las propiedades de un elemento del fichero abierto *ENTONCES* tengo que implementar un panel de propiedades en la columna derecha de la ventana principal.
Tareas	1. Añadir las nuevas clases y métodos de IPlugin a PluginAPI. 2. Implementar los cambios en XAMLPlugin. 3. Implementar los cambios en HTMLPlugin. 4. Obtener la posición seleccionada en el editor de texto. 5. Implementar un panel de propiedades en la columna derecha de la ventana principal. 6. Documentación. 7. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Ambos plugins deben implementar los métodos necesarios para obtener las propiedades. 3. El panel de propiedades debe mostrar las propiedades expuestas por los plugins.

Tabla 5.24: Historia de usuario n.º 27

En esta historia de usuario se incluyen todas las tareas necesarias para completar la implementación del panel de propiedades. Esta historia de usuario es demasiado grande como para poder estimar con precisión el tiempo que se tardará en total en completar, así que se añadirán nuevas historias de usuario al *backlog* para dividirla en tareas más sencilla. Estas historias de usuario contarán con sus propias tares y criterios de aceptación, y contribuirán en conjunto a alcanzar el objetivo de este Sprint. A continuación se comentarán las completadas en este Sprint.

Definir clase propiedad base	
Descripción	*COMO* desarrollador *QUIERO* tener una clase base para las propiedades de los elementos del visualizador *ENTONCES* hay que definir una clase para las propiedades con los métodos y campos necesarios.
Tareas	1. Definir la clase base. 2. Añadirla a la API de plugins. 3. Documentación. 4. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. PluginAPI debe contenr una clase base para las propiedades.

Tabla 5.25: Historia de usuario n.º 31

Dado que puede haber distintos tipos de propiedades según sus valores, se puede simplificar su manejo creando una clase base. La clase *VisualizerElementProperty* será abstracta y genérica, y contará con los métodos para obtener y modificar el valor de la propiedad, y la línea y la columna donde se declara el valor de la propiedad en el fichero. También se añadirá un evento que deberá ser invocado cada vez que se modifique el valor de la propiedad. En este evento se incluirá el nuevo valor de la propiedad y la posición (línea y columna) de la propiedad.

Implementar obtención de propiedades en IPlugin	
Descripción	*COMO* desarrollador *QUIERO* que los plugins tengan los métodos necesarios para manejar las propiedades *ENTONCES* tengo que implementar los métodos necesarios en la interfaz IPlugin.
Tareas	1. Añadir los métodos a IPlugin 2. Documentación. 3. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Los nuevos métodos deben permitir obtener las propiedades de un elemento del fichero.

Tabla 5.26: Historia de usuario n.º 32

En esta tarea se debían añadir los métodos para obtener las propiedades de un elemento a la API de plugins. Sin embargo no fue posible usar la clase *VisualizerElementProperty* con genéricos ya que esto obliga a utilizar métodos estáticos y a indicar en dichos métodos el tipo usado para el genérico. Por tanto, antes de añadir los nuevos métodos a la interfaz *IPlugin* se modificó la clase base de propiedades para que no utilice genéricos. En su lugar se utilizará la clase *object*, de la cual heredan todos los tipos en C#, y se indicará el tipo del valor con una enumeración que contendrá los tipos disponibles.

Una vez se han realizado estos cambios se pueden añadir los métodos necesarios a la API de plugins para obtener las propiedades. Además de esto también se creará un tipo de excepción específica para que los plugins indiquen cuando no es posible encontrar un elemento concreto.

Al final del Sprint no se llegó a completar el objetivo fijado, sólo se consiguió añadir las nuevas clases y métodos a la API de plugins, no se diseñó el panel de propiedades ni se implementaron los cambios en los plugins. Esto se debió a que no se tuvo en cuenta inicialmente la complejidad de la implementación del panel de propiedades con lo que la estimación temporal fue incorrecta. También se tuvo que atender otros compromisos que impidieron completar las tareas a tiempo.

5.7 Sprint 7

Durante este Sprint se completará el trabajo restante del anterior para implementar el panel de propiedades. El Sprint comenzó el 26/05/2019 y finalizó el 17/06/2019. Las historias de usuario realizadas fueron las siguientes:

Implementar obtención de propiedades en XAMLPlugin	
Descripción	*COMO* desarrollador *QUIERO* que XAMLPlugin permita la obtención de propiedades *ENTONCES* tengo que implementar los nuevos métodos de IPlugin para la obtención de propiedades.
Tareas	1. Definir nuevas clases derivadas de *VisualizerElementProperty*. 2. Implementar el nuevo método de obtención de propiedades de IPlugin. 3. Unit testing. 4. Documentación. 5. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. XAMLPlugin debe permitir la obtención de propiedades de controles definidos en ficheros XAML.

Tabla 5.27: Historia de usuario n.º 33

Con la primera tarea se crearon nuevas clases para propiedades con valores de tipo entero, de coma flotante, booleano y de texto. Después se añadió al bucle de lectura de ficheros XAML un diccionario en el que se guarda la posición de cada control definido en el fichero. De este modo, cuando se solicite las propiedades de un control según su posición tan sólo es necesario consultar dicho diccionario.

Para obtener las propiedades de un control concreto se obtendrán todos sus propiedades de dependencia, y después se proporcionarán aquellas que no sean adjuntas, no sean de sólo lectura, y que tengan visibilidad en la serialización. De este modo también se obtendrán incluso las propiedades a las que no se les haya asignado un valor en el fichero original. Con cada una de estas propiedades se creará una nueva clase derivada de *VisualizerElementProperty* y se devolverá la lista completa.

Implementar obtención de propiedades en HTMLPlugin	
Descripción	*COMO* desarrollador *QUIERO* que HTMLPlugin permita la obtención de propiedades *ENTONCES* tengo que implementar los nuevos métodos de IPlugin para la obtención de propiedades.
Tareas	1. Obtener un elemento HTML a partir de su posición. 2. Obtener los atributos del elemento. 3. Implementar GetProperties para obtener la lista. 4. Unit testing. 5. Documentación. 6. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. HTMLPlugin debe permitir la obtención de propiedades de controles definidos en ficheros HTML.

Tabla 5.28: Historia de usuario n.º 35

En el caso de HTML no existe una librería propia de .NET que permita analizar un fichero para determinar la posición de cada elemento y sus atributos. Por ello se optó por usar la librería AngleSharp, la cual sigue la especificación del *World Wide Web Consortium* para HTML. Con esta librería se puede obtener la posición de cada elemento durante la lectura del fichero, de modo que se puede proceder del mismo modo que con XAMLPlugin, añadiendo a un diccionario una referencia a cada elemento según su posición. Para obtener los atributos de un elemento también se puede utilizar también la librería AngleSharp, aunque hay que tener en cuenta que todos tendrán un valor de texto.

Obtener posición del cursor en el editor	
Descripción	*COMO* desarrollador *QUIERO* obtener la línea y la columna del cursor en el editor de texto *ENTONCES* implementar los nuevos métodos para obtener la posición del cursor.
Tareas	1. Cambiar el control a AvalonEdit. 2. Obtener el la línea y la columna del cursor. 3. Documentación. 4. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Se debe obtener la posición del cursor cuando se cambia su posición.

Tabla 5.29: Historia de usuario n.º 36

El diseño inicial de la aplicación en el Mockup incluye un editor de texto en la columna izquierda que muestre los números de líneas. El control *TextBox* usado hasta ahora permite editar el texto del fichero cargado, pero no funciona correctamente en cuanto a determinar el número de línea. Esto se debe a que cuando una línea del texto es demasiado larga y no cabe en el control se salta a la siguiente línea, pero con esto se incrementa también el número de líneas que proporciona el control.

Este comportamiento no resulta apropiado en este caso, por lo que se pasó a utilizar el control AvalonEdit. Este control si cuenta con la capacidad de mostrar los números de línea correctamente y de obtener la posición del cursor en el texto. Para ello tan sólo es necesario subscribirse a un evento que se activará cada vez que el usuario desplace el cursor en el editor de texto. El aspecto del control puede observarse en la figura 5.14 de la derecha.

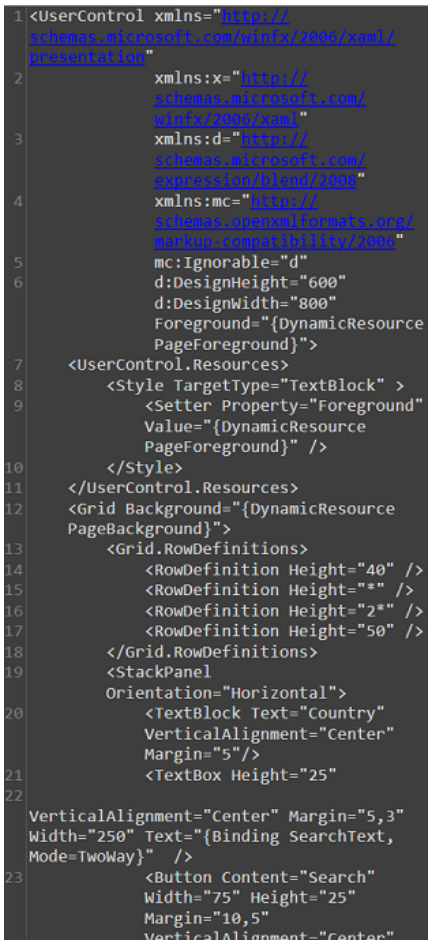

Fig. 5.14. Editor de texto en la versión 0.6.5

Implementar control de edición de propiedades	
Descripción	*COMO* usuario *QUIERO* ver y editar las propiedades de un elemento *ENTONCES* hay que diseñar un control para mostrar las propiedades.
Tareas	1. Diseñar un control para cada tipo de propiedad. 2. Añadir un panel en la columna derecha de la ventana principal. 3. Añadir las propiedades al panel cuando se seleccione un elemento. 4. Documentación. 5. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Al seleccionar un control se deben mostrar sus propiedades en el panel. 3. Cada control debe permitir solo los valores correctos según el tipo de propiedad.

Tabla 5.30: Historia de usuario n.º 37

Para cada tipo de propiedad comentado antes (entero, de coma flotante, booleano y de texto) se creará un control que permita editar su valor. Al igual que antes se implementará una clase base de la cual heredarán las demás, aunque en este caso sí se podrá usar un tipo genérico. Esta clase contará con los métodos para obtener y modificar el valor de la propiedad, pero estos podrán ser sobrescritos en las clases derivadas.

En la columna derecha se añadirá un panel que permita desplazarse verticalmente por todos las propiedades añadidas. El proceso de obtener las propiedades y añadirlas al panel se muestra en el diagrama de la figura 5.16, y el aspecto del panel de propiedades tras haber seleccionado un elemento en la figura 5.15.

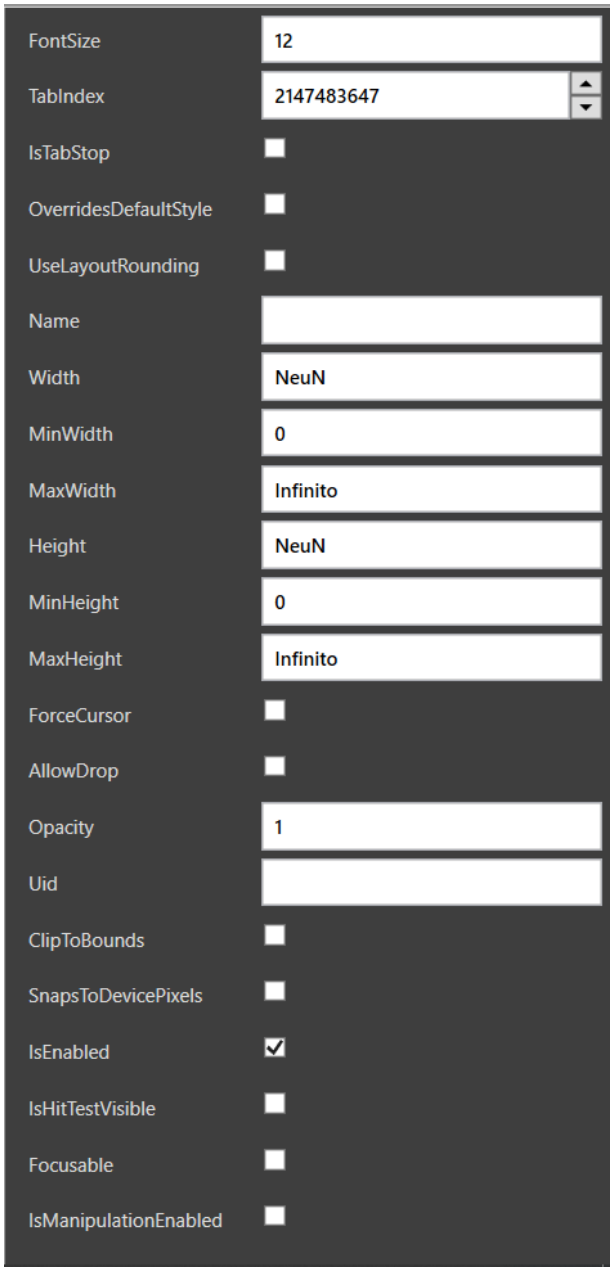

Fig. 5.15. Panel de propiedades en la versión 0.6.6

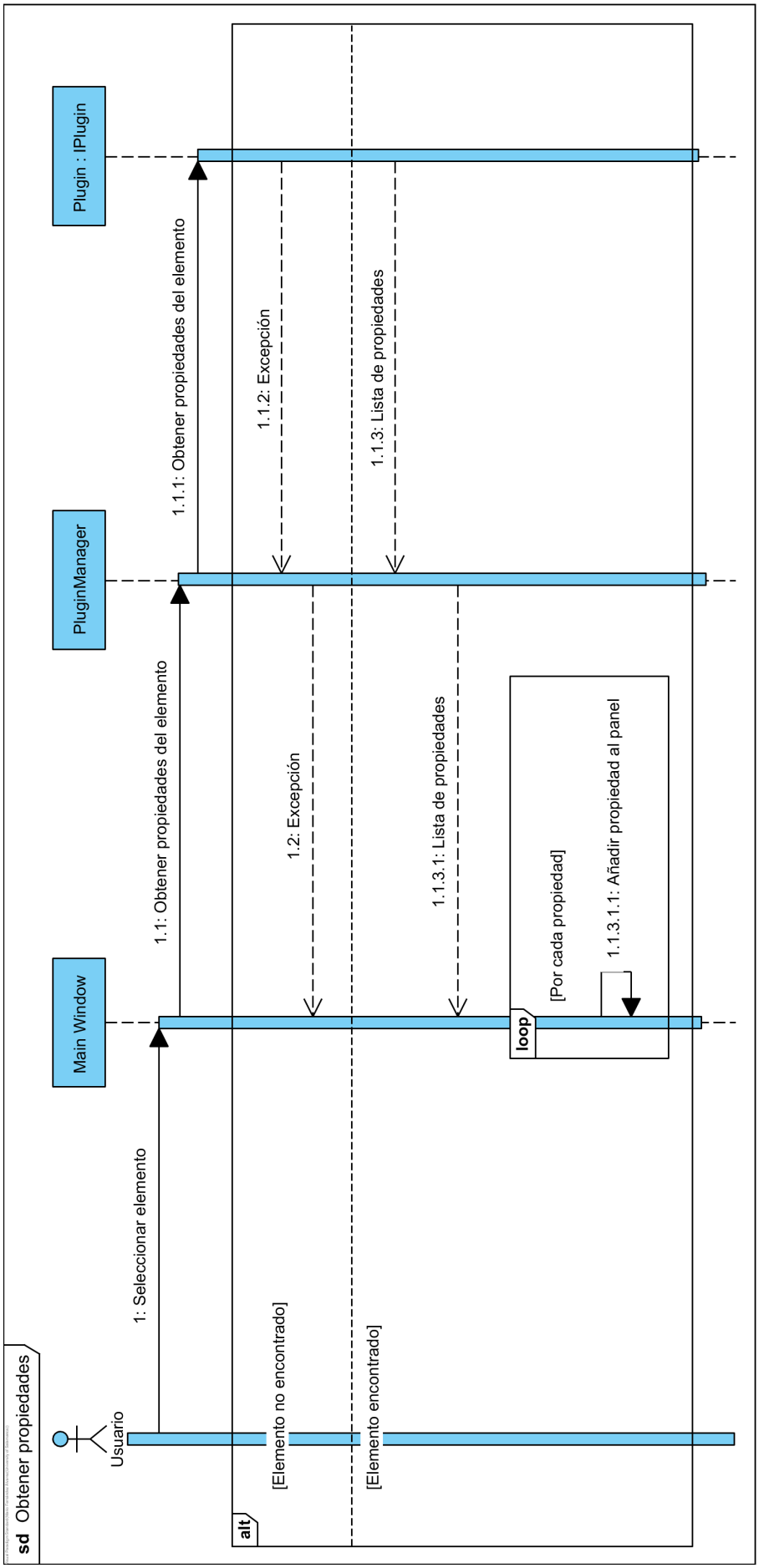

Fig. 5.16. Obtención de propiedades en la versión 0.6.6

En la revisión del Sprint se demostró el funcionamiento del panel de propiedades, el cual permite editar correctamente las propiedades de un elemento. Los nuevos valores se aplican correctamente al visualizador, pero no se modifican en el texto mostrado en la columna izquierda. En este Sprint se pudo alcanzar el objetivo fijado completando todas las tareas, lo cual supone una mejora clara frente al anterior.

5.8 Sprint 8

En este último Sprint el objetivo es implementar la función de guardar las modificaciones realizadas en el fichero cargado. También se realizarán algunas tareas menores para mejorar la interfaz. El Sprint durará solo 2 semanas, comenzará el 17/06/2019 y finalizará el 28/06/2019. Las historias de usuario ha completar son las siguientes:

Implementar guardado	
Descripción	*COMO* usuario *QUIERO* poder guardar las modificaciones realizadas *ENTONCES* tengo que implementar el guardado del fichero.
Tareas	1. Añadir nuevos métodos a PluginAPI. 2. Implementar los métodos en cada plugin. 3. Añadir los métodos al gestor de plugins. 4. Añadir botón guardado en la ventana principal. 5. Unit testing. 6. Documentación. 7. Merge request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Cada plugin debe permitir guardar el fichero. 3. El fichero guardado debe contener las nuevas modificaciones realizadas.

Tabla 5.31: Historia de usuario n.º 38

Poder guardar los cambios realizados es uno de los requisitos fundamentales de cualquier editor, por lo que no podía faltar en UIComposer. En esta historia de usuario se implementa esta funcionalidad en cada plugin, de modo que esta tarea se delegará a los plugins para que elijan la forma apropiada de guardar el contenido del fichero.

Una vez se hayan guardado los cambios en el fichero se deberá actualizar el visualizador para que estos se vean reflejados. Los plugins deberán proporcionar de nuevo este visualizador para mostrarlo en la ventana principal. El proceso se puede comprobar en el diagrama UML de secuencia 5.17.

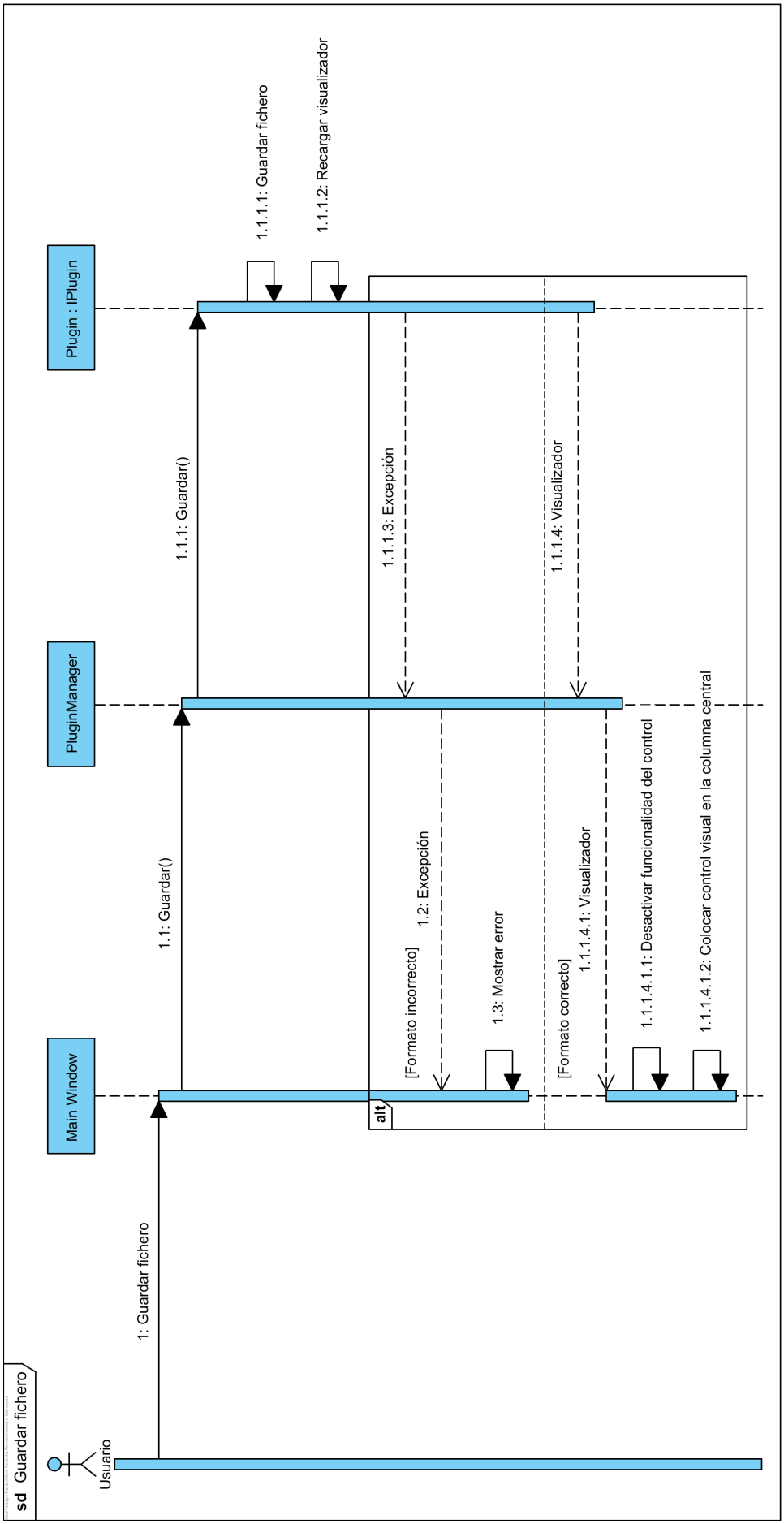

Fig. 5.17. Guardado fichero en versión 0.8.1

Siguiendo este procedimiento los cambios realizados se mostrarán tras haber guardado los cambios en el fichero. Este no era el comportamiento deseado inicialmente, lo ideal sería que los cambios se viesen reflejados en tiempo real en el visualizador sin necesidad de guardar el fichero. Esto no ha sido posible debido a que para evitar los problemas de rendimiento sería necesario aplicar los cambios en el visualizador en un hilo distinto al de la interfaz, lo cual no es recomendable tanto por rendimiento como por seguridad en WPF.

Aún así se permiten deshacer los cambios realizados tras guardar un fichero, de modo que si el aspecto final no es el deseado se puede deshacer y guardar de nuevo para volver a la versión anterior.

Resaltar en editor texto el elemento seleccionado	
Descripción	*COMO* ususario *QUIERO* ver el elemento seleccionado en el editor *ENTONCES* resaltar la posición del cursor en el editor de texto.
Tareas	1. Modificar las opciones del editor 2. Documentación. 3. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. En el editor se debe destacar la posición del cursor.

Tabla 5.32: Historia de usuario n.º 39

Como ya se explicó anteriormente el proceso de obtención de las propiedades comienza cuando el usuario selecciona un elemento en una línea concreta en el editor de texto. Con esta historia de usuario se modifica el editor para que resalte la línea donde se encuentra el cursor, con lo cual el usuario podrá seleccionar el elemento que desee más fácilmente.

Añadir atajos de teclado	
Descripción	*COMO* ususario *QUIERO* utilizar atajos de teclado para los comandos principales *ENTONCES* hay que añadir los comandos a los menús.
Tareas	1. Añadir comandos para cada acción. 2. Añadir los atajos de teclado. 3. Documentación. 4. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. Se deben poder ejecutar las acciones de guardar, salir, deshacer y rehacer con atajos de teclado.

Tabla 5.33: Historia de usuario n.º 40

Las acciones más comunes al trabajar con un editor son guardar los cambios, deshacerlos, rehacerlos y salir de la aplicación, por lo que proporcionar atajos de teclado para realizarlas

acelera el ritmo de trabajo. Rehacer y deshacer los cambios se incluye directamente en el control AvalonEdit, mientras que para guardar y salir se asocia cada entrada del menú correspondiente con la acción con una combinación de teclas. Las combinaciones son:

- **Guardar** = Control + S
- **Deshacer** = Control + Z
- **Rehacer** = Control + Y
- **Salir** = Control + Q

Estas combinaciones han sido las elegidas porque son las más habituales para estas acciones en la mayoría de aplicaciones. También se mostrarán en los menús para poder verlas directamente, con lo que su aspecto final será el de las figura 5.18 y 5.19.

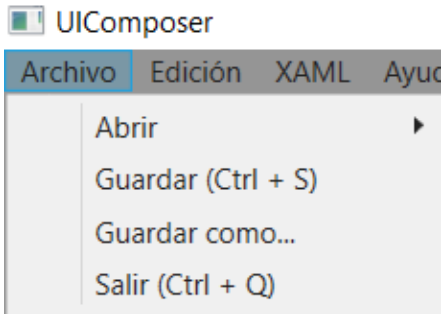

Fig. 5.18. Menú archivo en la versión 0.8.3.

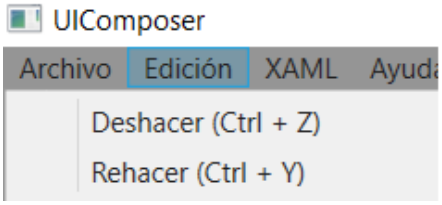

Fig. 5.19. Menú edición en la versión 0.8.3.

Añadir licencias software	
Descripción	*COMO* desarrollador *QUIERO* incluir las licencias del software usado en la aplicación y los plugins *ENTONCES* hay que añadir las licencias de AvalonEdit y AngleSharp.
Tareas	1. Añadir nuevo menú ayuda en la ventana principal. 2. Añadir licencia de AvalonEdit al menú de ayuda. 3. Añadir licencia de AngleSharp al menú de HTMLPlugin. 4. Documentación. 5. Pull request.
Criterios de aceptación	1. Los resultados deben de estar en la Wiki. 2. La licencia de AvalonEdit debe incluirse con la aplicación UIComposer. 3. La licencia de AngleSharp debe incluirse con el plugin de HTML.

Tabla 5.34: Historia de usuario n.º 41

AvalonEdit y AngleSharp son proyectos distribuidos bajo la licencia de software libre del MIT (*Massachusetts Institute of Technology*), la cual permite usar y modificar el software libremente. Esta licencia requiere únicamente que esta y los derechos de autor sean incluidos en las copias del software, por lo que deben ser incluidas en la aplicación y en el plugins.

Para cumplir con estas licencias se creará un botón de ayuda en el menú superior que contendrá una entrada con el título "Acerca de UIComposer". Al pulsar este botón se abrirá una nueva ventana donde se mostrará una descripción de la aplicación, la versión actual y la licencia de AvalonEdit.

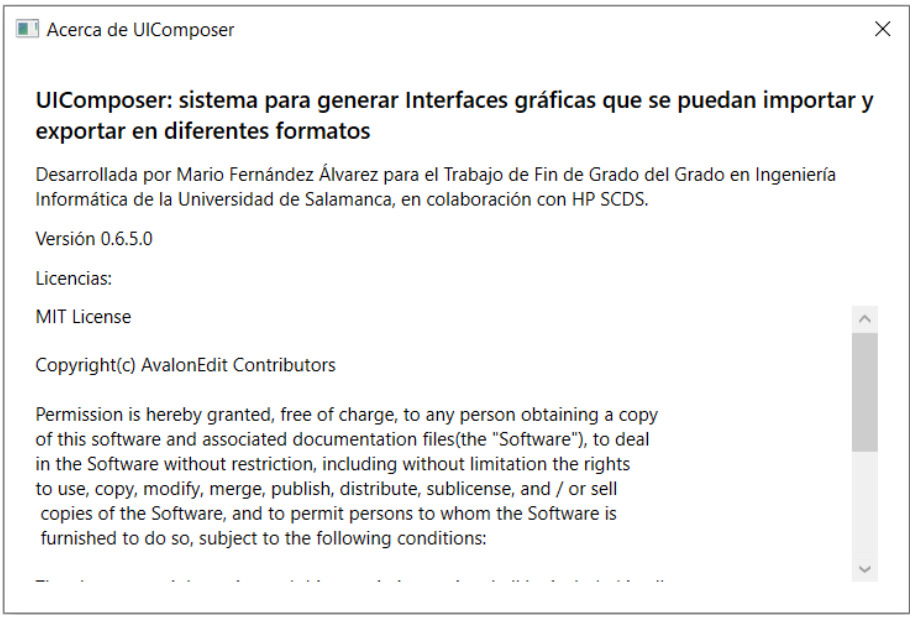

Fig. 5.20. Ventana acerca de UIComposer en la versión 0.8.4

En el caso de AngleSharp se procederá del mismo modo, añadiendo una nueva entrada en el menú del plugin HTML con la que se abra una nueva ventana donde se muestre la licencia de AngleSharp.

En la revisión del Sprint se pudo comprobar que la función de guardado estaba implementada y cumple con el objetivo establecido. También se valoraron positivamente los cambios realizados en la interfaz, tanto el resaltado de línea como los atajos de teclado.

En la reunión de revisión del Sprint se comprobó el correcto funcionamiento del guardado de archivos y de los atajos de teclado. En cuanto a la retrospectiva se valoró que pese a ser un Sprint más corto las tareas se completasen a tiempo.

5.8.1 Revisión final

Al ser este el último Sprint también se evaluó el trabajo global completado hasta ahora. A lo largo del desarrollo se ha seguido el modelo iterativo e incremental de la metodología Scrum, construyendo las bases del proyecto al inicio y añadiendo nuevas funcionalidades y mejorando las ya existentes a medida que se avanzaba. Gracias a esto se ha conseguido evolucionar progresivamente tanto en la aplicación principal como en los plugins, manteniendo un control cercano sobre los cambios realizados y su validación.

El diseño resultante cumple con los estándares de calidad esperados, especialmente dada la complejidad de diseñar una arquitectura extensible que permita compatibilizar tecnologías de dominios de aplicación distintos. En la arquitectura se puede comprobar claramente la evolución del proyecto si se compara la versión inicial representada en la figura 5.4 con la versión final de la figura 5.21.

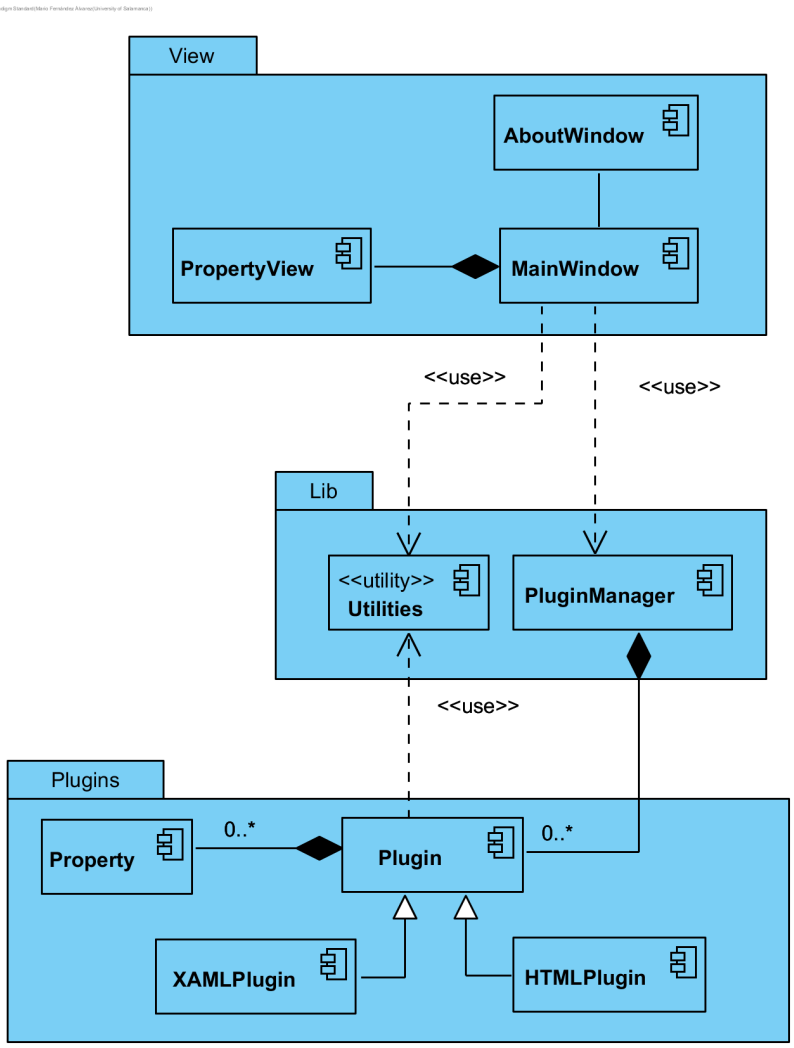

Fig. 5.21. Arquitectura final.

Como se puede ver el sistema *software* ha mantenido la organización estructural inicial con los tres paquetes de vista, lógica y plugins, pero se han añadido nuevos componentes y se ha refinado su interacción. Los plugins proporcionarán los servicios definidos en la API, y estos podrán ser solicitados por la vista a través del gestor. El manejo de los plugins es por tanto tarea exclusiva del gestor, que ha pasado a ser la pieza central del sistema.

De este modo se ha conseguido eliminar las dependencias directas entre la vista y los plugins, limitando así su influencia sobre la implementación de los componentes de la capa de presentación. Esto ha supuesto una ventaja clara en el desarrollo al poder trabajar paralelamente en añadir las nuevas funcionalidades tanto a los plugins como a la aplicación principal.

La interacción de estos componentes y las clases concretas del sistema se pueden observar con mayor detalle en los anexos de análisis y diseño.

6 Producto

Tras finalizar el desarrollo del proyecto se puede proceder a comprobar los frutos del trabajo, es decir, el producto final. En este apartado se mostrarán las características y funcionalidades más importantes de la aplicación usando como ejemplo un fichero con formato XAML.

6.1 Inicio

Una vez se haya descargado y extraído el paquete para ejecutar la aplicación no se requiere ninguna configuración ni instalación previa, tan solo hace falta ejecutar fichero "UIComposer.exe". Al inicio se mostrará la ventana principal de la aplicación tal y como se puede ver en la figura 6.1.

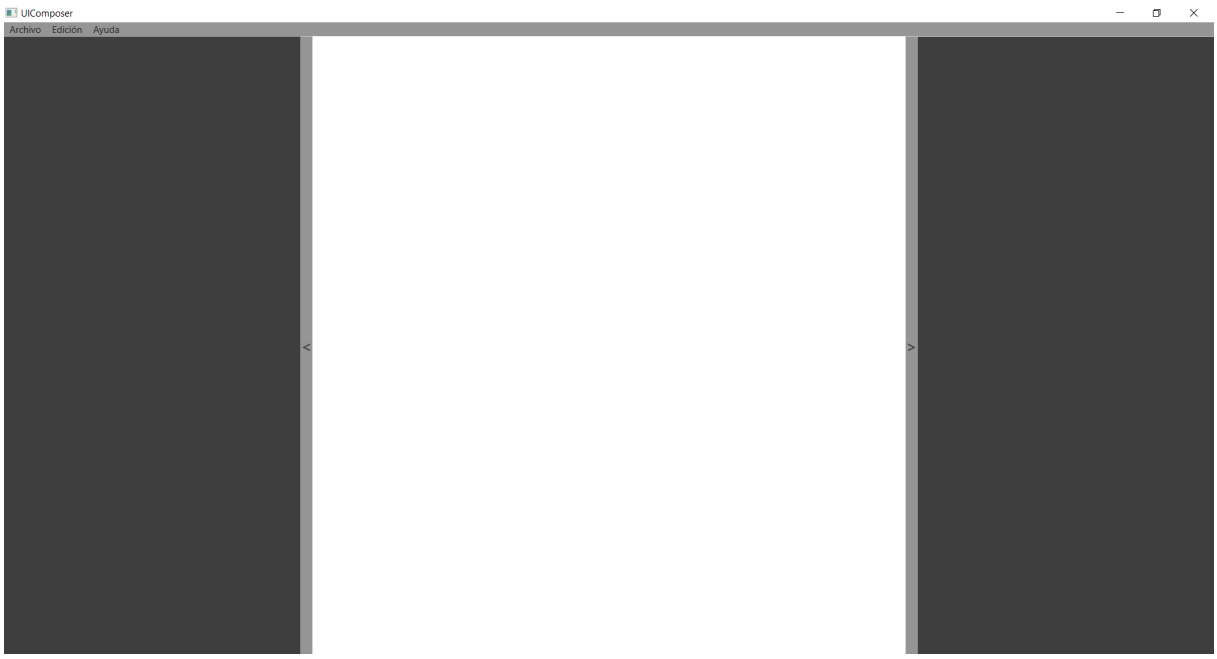

Fig. 6.1. Ventana principal.

Las tres columnas principales con el editor de texto, el visualizador y el panel de propiedades se mostrarán vacías ya que aún no se ha cargado un fichero. Estas se podrán redimensionar utilizando las guías que aparecen a los lados de la columna central con un color gris claro

6.2 Abrir fichero

Para abrir un fichero se deberá desplegar el menú “Fichero” de la barra superior y elegir el formato deseado en la opción “Abrir” (6.3). En este caso se abrirá el fichero “UserControl.xaml” que se puede encontrar en la carpeta “docs/samples” del repositorio.

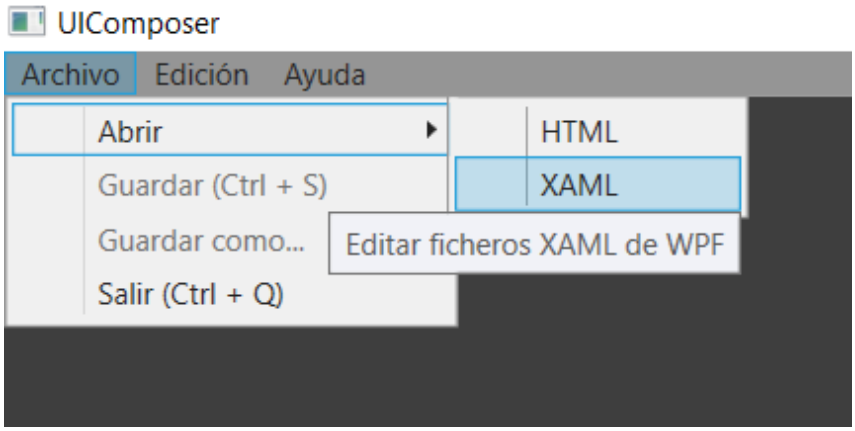

Fig. 6.2. Abrir fichero XAML.

Tras pulsar esta opción se mostrará la ventana de apertura de ficheros propia de Windows (figura 6.3). El filtro de la ventana para la extensión de ficheros será el indicado por el plugin, y en este caso será “.xaml”.

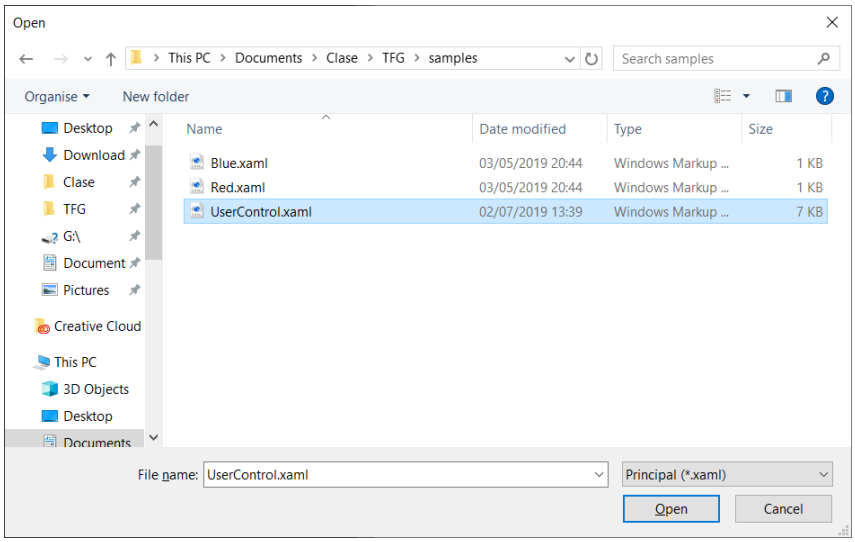

Fig. 6.3. Ventana apertura fichero.

6.3 Fichero abierto

Cuando se cargue el fichero se mostrará su contenido en el editor de la columna izquierda y el visualizador obtenido del plugin aparacerá en la columna central (figura 6.4).

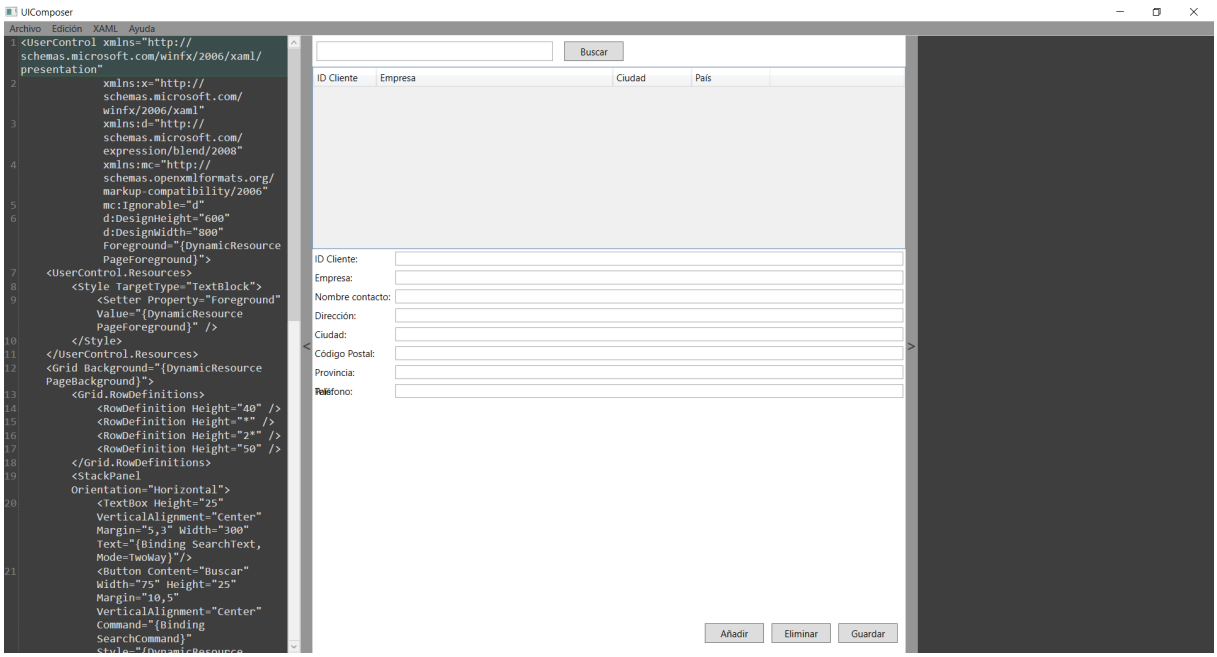

Fig. 6.4. Ventana principal tras cargar fichero XAML.

6.4 Editor de texto

```
<UserControl xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             mc:Ignorable="d"
             d:DesignHeight="600"
             d:DesignWidth="800"
             Foreground="{DynamicResource PageForeground}">
    <UserControl.Resources>
        <Style TargetType="TextBlock">
            <Setter Property="Foreground" Value="{DynamicResource PageForeground}" />
        </Style>
    </UserControl.Resources>
    <Grid Background="{DynamicResource PageBackground}">
        <Grid.RowDefinitions>
            <RowDefinition Height="40" />
            <RowDefinition Height="*" />
            <RowDefinition Height="2*" />
            <RowDefinition Height="50" />
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal">
            <TextBox Height="25" VerticalAlignment="Center" Margin="5,3" Width="300" Text="{Binding SearchText, Mode=TwoWay}"/>
            <Button Content="Buscar" Width="75" Height="25" Margin="10,5" VerticalAlignment="Center" Command="{Binding SearchCommand}"
            Style="{DynamicResource
```

Fig. 6.5. Editor de texto.

En el editor de texto se mostrará el contenido del fichero indicando el número de línea a la izquierda del texto, y también se destacará la línea donde se encuentre el cursor.

Usando el teclado se puede modificar directamente el texto como en cualquier otro editor, y además se podrán deshacer y rehacer los cambios con los atajos de teclado o el menú de edición.

Si el texto ocupa más tamaño que la columna se podrá desplazar verticalmente con la barra de desplazamiento o la rueda del ratón.

6.5 Visualizador

En la columna central se mostrará el visualizador generado por "XAMLPlugin" tras analizar el contenido del fichero. La interfaz que aparece en la figura 6.6 representa un control de usuario con varios controles como paneles, cuadrículas, botones, bloques de texto, etc. Para poder ver los cambios realizados en el fichero original es necesario guardarlos antes. Una vez se guarde el visualizador será recargado por el plugin.

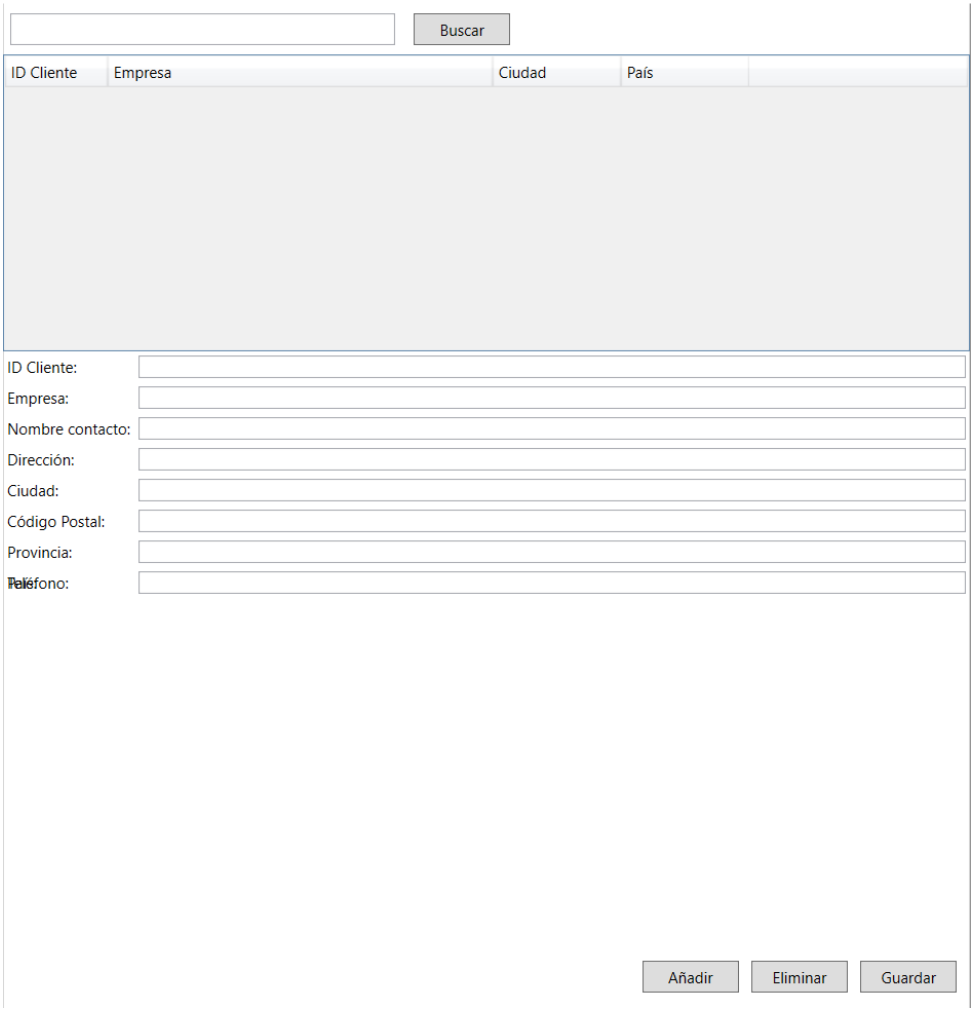

Fig. 6.6. Visualizador XAML.

6.6 Panel de propiedades

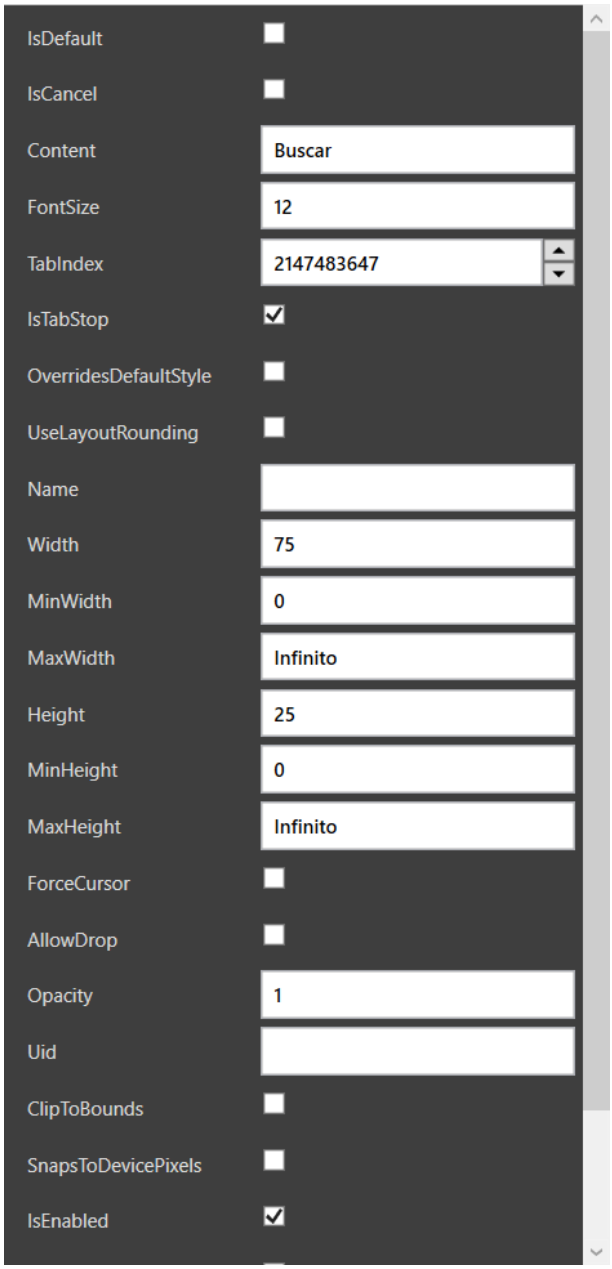

Fig. 6.7. Panel de propiedades.

El panel de propiedades contendrá las vistas de cada una de las propiedades del elemento definido en la línea seleccionada en el editor. Para cada propiedad se mostrará una etiqueta con su nombre y un control específico para modificar su valor.

- Los valores booleanos (verdadero o falso) se editarán con una casilla de verificación.
- Los valores de texto se editarán con un cuadro en el que se podrá indicar cualquier valor por teclado.
- Los valores de números enteros se editarán con un cuadro en el que se mostrará el valor númerico. Este valor podrá modificarse con el teclado o con las flechas de la parte derecha para incrementar o disminuir en una unidad.
- Los valores de números decimales se podrán editar únicamente por teclado en el recuadro indicado. En este caso si la propiedad no contiene un valor concreto inicialmente se mostrará "Infinito".

7 Conlusiones

Para concluir la memoria en esta sección se evaluará el producto final contrastando los objetivos planteados inicialmente, y también se propondrán algunas mejoras posibles para mejorar la funcionalidad de la aplicación..

El objetivo principal del proyecto se ha conseguido ya que la aplicación UIComposer permite editar y visualizar ficheros de interfaces gráficas de usuario con distintos formatos. La herramienta cuenta con los 3 componentes esenciales que se habían establecido:

- El editor de texto permite modificar directamente el contenido del fichero, y estos cambios pueden guardarse sobreescribiendo el fichero original o en uno nuevo. Además cumple con el diseño establecido en el Mockup mostrando los números de línea.
- El visualizador proporcionado por XAMLPlugin y HTMLPlugin muestra la interfaz definida en el fichero, y también refleja los cambios realizados con el editor de texto y con el panel de propiedades.
- El panel de propiedades permite editar cada una de las propiedades del elemento seleccionado en el editor de texto. Esto es posible gracias a que cada plugin puede identificar cada elemento según su posición y así obtener sus propiedades.

En cuanto a los objetivos de integración y despliegue continuo se han producido algunos problemas durante el desarrollo del proyecto, pero en general se puede decir que estos procesos se han aplicado correctamente. Su uso ha resultado especialmente positivo a la hora de detectar errores, ya que, en las ocasiones en que estos no han sido detectados a tiempo durante la implementación, las pruebas unitarias ejecutadas en la integración han sido efectivas.

Con la aplicación UIComposer se ha materializado la idea inicial de construir una herramienta extensible que permita trabajar con distintos formatos, de modo que el trabajo realizado ha sido un éxito. Aún así existen varios puntos en los que caben varias mejoras posibles:

- Los visualizadores cumplen con su función, pero la representación de los elementos se puede ver deformada al ocupar toda la columna central. Limitar el tamaño de los visualizadores dentro de un recuadro concreto en la columna resultaría más apropiado. Además se podría controlar el tamaño de este recuadro tal y como se propuso originalmente en el Mockup.
- Continuando con los visualizadores, también sería de gran ayuda el poder interactuar con los elementos de manera gráfica. Por ejemplo, cambiando su tamaño y su posición arrastrando un elemento con el ratón.
- El editor de texto empleado cuenta con la posibilidad de definir un patrón de colores personalizado, pero no se ha llegado a implementar el resaltado de sintaxis en cada plugin para cada lenguaje.

Referencias

Beck, K. (1989). *Simple Smalltalk Testing : With Patterns*. doi:10.1017/CBO9780511574979.033

Beck, K., Grenning, J., Martin, R. C., Beedle, M., Highsmith, J., Mellor, S., ... Marick, B. (2001). *Manifesto for Agile Software Development*. Agile Alliance. Recuperado desde http://agilemanifesto.org/

Caldwell, B., Cooper, M., Guarino Reid, L. & Vanderheiden, G. (2008). *Web Content Accessibility Guidelines (WCAG) 2.0* (inf. téc. N.º 2009-11-23). Recuperado desde http://www.w3.org/TR/WCAG20/

García-Peñalvo, F. J. & Vázquez-Ingelmo, A. (2019). Metodologías de Ingeniería de Software. doi:10.5281/ZENODO.2561341

Gladwell, M. (2011). Creation Myth. *The New Yorker*. Recuperado desde https://www.newyorker.com/magazine/2011/05/16/creation-myth

Gutierrez Miranda, M. (2011). The importance of graphic users interface, analysis of graphical user interface design in the context of human-computer interaction. En *EDULEARN11 Proceedings* (pp. 7137-7144). 3rd International Conference on Education and New Learning Technologies. IATED.

Hartman, B. (2009). New to agile? INVEST in good user stories. Recuperado el 13 de junio de 2019, desde https://agileforall.com/new-to-agile-invest-in-good-user-stories/

IEEE. (1998). *IEEE 830-1998: IEEE Recommended Practice for Software Requirements Specifications*. doi:10.1109/IEEESTD.1998.88286

Loukides, M. (2017). The evolution of DevOps. Recuperado el 13 de junio de 2019, desde https://medium.com/@mikeloukides/the-evolution-of-devops-9d101124aa2e

Microsoft. (2017a). Documenting your code with XML comments. https://docs.microsoft.com/es-es/dotnet/csharp/codedoc. Recuperado el 23 de junio de 2019, desde https://docs.microsoft.com/es-es/dotnet/csharp/codedoc

Microsoft. (2017b). Understanding XAML Node Stream Structures and Concepts. Recuperado el 22 de junio de 2019, desde https://docs.microsoft.com/https://docs.microsoft.com/en-us/dotnet/framework/xaml-services/understanding-xaml-node-stream-structures-and-concepts

Preston-Werner, T. (2013). Versionado Semántico 2.0.0. https://semver.org/lang/es/. Recuperado el 14 de junio de 2019, desde https://semver.org/lang/es/

Santos Luaces, P. (2019). *Version Control, DevOps and Agile Development with Plastic SCM: The Plastic SCM book*. Recuperado desde https://www.plasticscm.com/book/

Schwaber, K. (2004). *Agile Project Management with Scrum*. Best practices. Microsoft Press.

Thacker, C., McCreight, E., Lampson, B., Sproull, R. & Boggs, D. (1979). *Alto: A personal computer*. Recuperado desde https://www.microsoft.com/en-us/research/publication/alto-personal-computer/

Wired Staff. (2004). The Click Heard Round The World. *Wired*. Recuperado desde https://www.wired.com/wired/archive/12.01/mouse.html