

**U**niversidade

de **V**igo

**E**scola **S**uperior de **E**nxeñaría **I**nformática

|  |
| --- |
| Memoria do Traballo de Fin de Grao que presenta  **D. Daniel Camba Lamas**  para a obtención do Título de Graduado en Enxeñaría Informática.  ***Editor visual para la generación de archivos XML con la descripción del comportamiento interactivo de un prototipado falso.*** |

|  |  |
| --- | --- |
|  | Xuño, 2017  Traballo de Fin de Grao Nº:  **Titor/a:** Javier Rodeiro Iglesias  Área de coñecemento:  **Departamento:** |

A ***Manuel*** y ***Pastora***, por darme los medios y el cariño para llegar hasta aquí.

A ***Diego***, ***Héctor*** y ***Román*** por las noches en vela, las de estudio y las de copas.

A ***Alba*** por nunca dejar que me rindiera y el enorme apoyo que ha sido en mi vida.

4. INDICE DE CONTENIDOS

[4. INDICE DE CONTENIDOS 3](#_Toc481775586)

[5. INDICE DE ILUSTRACIONES 4](#_Toc481775587)

[6. INDICE DE TABLAS 5](#_Toc481775588)

[7. INTRODUCCIÓN 6](#_Toc481775589)

[8. OBJETIVOS 6](#_Toc481775590)

[9. RESUMEN DE LA SOLUCIÓN PROPUESTA 7](#_Toc481775591)

[10. PLANIFICACIÓN Y SEGUIMIENTO 7](#_Toc481775592)

[11. ARQUITECTURA 8](#_Toc481775593)

[11.1. Física 8](#_Toc481775594)

[11.2. Lógica 8](#_Toc481775595)

[11.3. Software 9](#_Toc481775596)

[12. TECNOLOGÍAS E INTEGRACIÓN DE PRODUCTOS DE TERCEROS 10](#_Toc481775597)

[13. ESPECIFICACIÓN Y ANÁLISIS DE REQUISITOS 10](#_Toc481775598)

[13.1. Requisitos funcionales 10](#_Toc481775599)

[13.2. Requisitos no funcionales 11](#_Toc481775600)

[14. DISEÑO DEL SOFTWARE (ESTÁTICO Y DINÁMICO) O DEL HARDWARE 11](#_Toc481775601)

[15. GESTIÓN DE DATOS Y INFORMACIÓN 11](#_Toc481775602)

[16. PRUEBAS 12](#_Toc481775603)

[17. MANUAL DE USUARIO 12](#_Toc481775604)

[18. PRINCIPALES APORTACIONES 12](#_Toc481775605)

[19. CONCLUSIONES 12](#_Toc481775606)

[20. VIAS FUTURAS DE TRABAJO 12](#_Toc481775607)

[21. REFERENCIAS 13](#_Toc481775608)

[22. APARTADO ADICIONALES 13](#_Toc481775609)

5. INDICE DE ILUSTRACIONES

6. INDICE DE TABLAS

7. INTRODUCCIÓN

Si pensamos por un momento cada una de las interfaces que tenemos a nuestro alcance a diario: La aplicación que usamos para leer noticias, el sistema operativo de nuestro móvil u ordenador, la botonera del coche o nuestro microondas. Todas ellas han pasado por un proceso de diseño (más o menos) riguroso, y en las fases de ese diseño nos encontramos con el *prototipado falso*, donde el diseñador de interfaces expone a los demás integrantes del equipo o al cliente, un esbozo de cómo funcionará la aplicación, la botonera de una máquina, etc.

Dicho esbozo será un conjunto de imágenes ordenadas, donde el diseñador nos expone las diferentes vistas y/o comportamientos que derivan de la interacción con dicha interfaz. Pero pese al orden detallado que pueda darnos el diseñador, la visualización que obtenemos de la interfaz resulta estática y por ello, surge este proyecto…

Este proyecto llevará a cabo la implementación de una parte del trabajo teórico llevado a cabo por el tutor de éste proyecto **Javier Rodeiro Iglesias** y que posteriormente fue continuado por **Pedro Miguel Teixeira Faria**, donde se define pormenorizadamente lo que es una interfaz, por qué está compuesta y como estos componentes interaccionan para crear dicha interfaz.

En el marco teórico citado se define que los componentes más básicos de una interfaz pueden definirse mediante dibujo o mediante imágenes, por lo que por motivos de alcance en este proyecto nos basaremos en las imágenes.

Así el objetivo será implementar un sistema de interacción con imágenes rasterizadas (BMP, JPG, PNG) siendo PNG el formato preferido debido al soporte de transparencias, donde el usuario sea capaz de definir elementos simples y complejos de la interfaz y la interacción entre ellos de la manera más intuitiva posible.

Maquetando así el *prototipado falso*, con sus diferentes vistas y comportamientos. Para obtener finalmente un archivo XML donde estará definida toda la interfaz como la interacción entre sus componentes, organizado de manera jerárquica.

Para que posteriormente dicho archivo pueda ser cargado en un visualizador que interpretará las etiquetas del XML a fin de otorgar una experiencia dinámica e interactiva sobre el *prototipado falso*. Sin embargo, dicho visualizador está fuera del alcance de este proyecto.

8. OBJETIVOS

* **OBJ-001:** Construir una aplicación gráfica de escritorio que resulte intuitiva.
* **OBJ-002:** Realizar operaciones básicas con imágenes: Mover y Escalar.
* **OBJ-003:** Gestión de componentes simples, basados en imágenes, los cuales agregan la necesidad de conocer:
  + Posición
  + Tamaño
  + Nivel de profundidad
  + Si es visible en la escena
  + Si está activo en la escena
* **OBJ-004:** Gestión de componentes complejos, formados por componentes simples u otros componentes complejos, los cuales:
  + Permiten definir estados, siendo un estado el conjunto de características de los componentes que lo componen.
  + Permiten definir los eventos que desencadenan las transiciones entre los estados. (A nivel de alcance se han limitado estos eventos a los eventos soportados por dispositivos táctiles)
  + Permiten definir si dichas transiciones tienen alguna precondición o postcondición, siendo una precondición la dependencia de que otro componente se encuentre en un estado determinado.
* **OBJ-005:** Que permita la manipulación visual de dichos componentes.
* **OBJ-006:** Que permita la manipulación textual de dichos componentes, mostrando una tabla con las características más importantes de cada componente y permita realizar las mismas acciones que desde el área de trabajo (*manipulación visual*) a excepción de mover y escalar, las cuales se consideran acciones únicamente visuales.
* **OBJ-007:** Que permita, en base a la especificación del marco teórico del que parte este proyecto:
  + Guardar el estado del prototipado en un archivo XML donde de manera jerárquica y con etiquetas específicas, se define con exactitud la interfaz en base a componentes, los estados de estos y las interacciones requeridas, para llevar a cabo las transiciones.
  + Cargar un archivo XML de forma que recupere el prototipado exactamente donde lo dejamos y poder así hacer modificaciones en los componentes que ya teníamos definidos o definir componentes nuevos.
* **OBJ-008:** Que permita visualizar su interfaz en varios idiomas.
* **OBJ-009:** Control del historial de acciones realizadas sobre los componentes para disponer de las acciones Deshacer y Rehacer.

9. RESUMEN DE LA SOLUCIÓN PROPUESTA

10. PLANIFICACIÓN Y SEGUIMIENTO

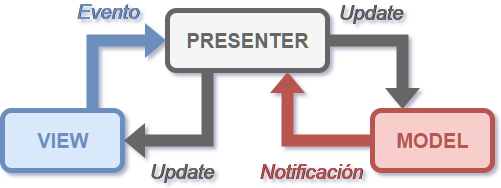
11. ARQUITECTURA

11.1. Física

Al tratarse de una aplicación donde la gestión de datos, visualización e interacciones no dependen de elementos externos, si no que se concentran en la máquina que la ejecuta; la arquitectura de la aplicación es *Stand-Alone*.

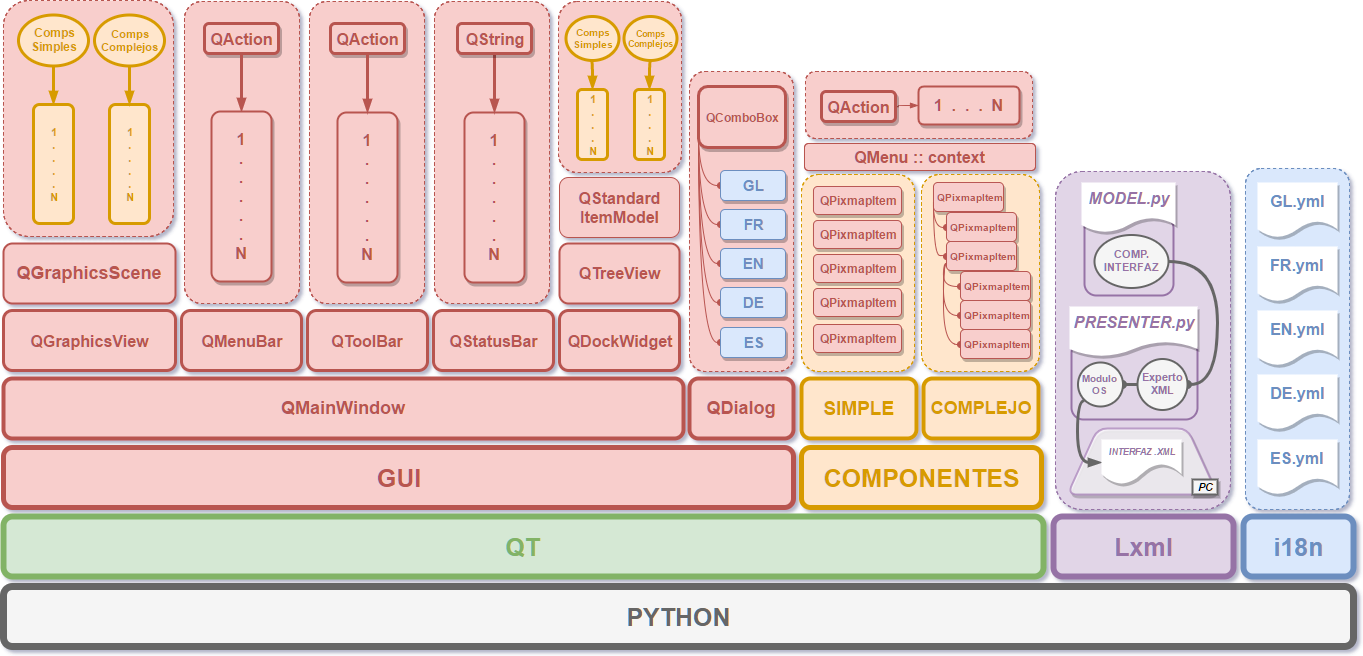
11.2. Lógica

Al tratarse de una aplicación para escritorio, con interfaz gráfica, la arquitectura utilizada será una arquitectura en capas siguiendo el patrón ***MVP con vista pasiva***; el cual es una evolución del archiconocido *MVC*.



En ***MVP con vista pasiva***, la ***V****ista* sólo define la interfaz de usuario, deposita toda la lógica de interfaz en el ***P****resentador* el cual posee la *lógica de negocio* y ejecuta la *lógica de interfaz* cuando se dispara un *Evento de usuario*. En base a ello, el ***P****resentador* actualiza el ***M****odelo* el cual una vez actualice la capa de persistencia, notificará al ***P****resentador* de los cambios y éste actualizará la ***V****ista.*

11.3. Software



**Figura 5.3.1.**  Diagrama de componentes, arquitectura de software.

12. TECNOLOGÍAS E INTEGRACIÓN DE PRODUCTOS DE TERCEROS

Tratándose de una aplicación con interfaz gráfica, resultaba obvio la necesidad de una librería o framework que cubriera dicha necesidad. Dado que la aplicación ha sido pensada para escritorio y el rendimiento de *Electron.io* es bajo, se descartó el uso de tecnologías web; por lo que los candidatos más populares eran GTK y Qt.

El desarrollo de la aplicación quería llevarse a cabo utilizando Python y aunque ambas tenían *bindings* para el lenguaje, se ha utilizado finalmente Qt por resultar (subjetivamente) más intuitivo. A mayores de que Qt provee de más elementos multiplataforma que sólo elementos de GUI como el tratamiento de hilos y sistema de ficheros, cosa que GTK no posee. Y esto hace que de querer actualizar la aplicación con nuevas funcionalidades complejas y multiplataforma, resulte más sencillo.

Respecto a la lógica de negocio y otras características de la interfaz que no pudieron resolverse por carencias de PyQt:

* **Python-i18n[YAML] (**<https://github.com/tuvistavie/python-i18n>**)**: Para leer archivos YAML con el texto de la interfaz en diferentes idiomas (Español, Inglés, Frances y Alemán)
* **Lxml (**<http://lxml.de>**)**: Para la persistencia de datos, ya que guardaremos y cargaremos el estado de un proyecto en un archivo XML.

13. ESPECIFICACIÓN Y ANÁLISIS DE REQUISITOS

13.1. Requisitos funcionales

El software debe poder…

* **RF-001:** Definir un prototipado nuevo.
* **RF-002:** Dar nombre al prototipado.
* **RF-003:** Guardar estado del prototipado en un fichero XML.
* **RF-004:** Carga el estado del prototipado a partir de un fichero XML con su definición.
* **RF-005:** Crear componentes simples.
* **RF-006:** Borrar componentes simples.
* **RF-007:** Dar nombre a un componente simple.
* **RF-008:** Modificar mediante un menú contextual parte de atributos del componente simple.
* **RF-009:** Modificar mediante gestos de ratón, los atributos relativos a posición (X e Y) y tamaño.
* **RF-010:** El software debe poder acceder al sistema de ficheros del usuario para cargar las imágenes.
* **RF-011:** El software debe poder Gestionar componentes complejos. Es decir […]
* **RF-012:** El software debe poder convertir objetos de Python a XML.
* **RF-013:** El software debe poder convertir XML a objetos Python.
* **RF-014:** Selección de idioma.
* **RF-015:** Acceso a historial de acciones mediante comandos de teclado.

13.2. Requisitos no funcionales

* **RNF-001:** Uso de estructuras de datos seguras en procesamiento paralelo.
* **RNF-002:** Soporte multi-idioma.
* **RNF-003:** Mantener consistencia respecto a otras herramientas de manipulación de imágenes.
* **RNF-004:** Funcionar off-line.
* **RNF-005:** Mostrar la última acción realizada, zoom del área de trabajo y, otros datos de interés en la barra de estado.
* **RNF-006:** Almacenar estado de la escena cada vez que se lleva a cabo una manipulación de los objetos que contiene, para soportar las acciones Deshacer y Rehacer.

14. DISEÑO DEL SOFTWARE (ESTÁTICO Y DINÁMICO) O DEL HARDWARE

15. GESTIÓN DE DATOS Y INFORMACIÓN

La aplicación es *Stand-Alone* y su persistencia se basa en un fichero por proyecto, por lo que no ha sido necesaria la utilización de bases de datos.

En tiempo de ejecución los datos son almacenados en objetos (*los* ***componentes***) los cuales se almacenan en una serie de colas dobles (***collections.deque*** *en la implementación*), ya que presentan una mayor eficiencia respecto a los arrays, estas colas definen las escenas (áreas visibles de la aplicación donde manejamos los elementos gráficos que definen a los componentes), siendo la escena creada a partir de los componentes presentes en estos contenedores.

Para la existencia de un historial (*Hacer/Deshacer*) se utilizan igualmente colas dobles, con la distinción de que éstas tienen una capacidad limitada, a fin de mantener en niveles coherentes el consumo de memoria RAM por parte de la aplicación. Ya que, para poder volver a un estado anterior o posterior de un prototipado, han de guardarse, por lo que ante cada modificación (textual o visual) en los atributos de un componente, antes de aplicar dicha modificación se guarda el estado del prototipado, es decir, guardamos una copia de la cola doble que define la escena, como un elemento de la cola doble que define el histórico de *hacer* o *deshacer*.

En persistencia, por la propia definición de ***componente*** en este proyecto, la relación entre componentes es jerárquica, por lo que en el momento de guardar el prototipado que se esté definiendo en un momento dado, nos valdremos del componente de mayor jerarquía para iterarlo y crear la definición del prototipado en un fichero XML con etiquetas propias.

Para realizar la carga de un prototipado, leeremos un fichero XML con una estructura que siga la especificación y construiremos los objetos pertinentes para manejar la definición del prototipado.

16. PRUEBAS

17. MANUAL DE USUARIO

18. PRINCIPALES APORTACIONES

* Agregar Z a la definición […]

19. CONCLUSIONES

20. VIAS FUTURAS DE TRABAJO

21. REFERENCIAS

22. APARTADO ADICIONALES