UNIVERSIDADE
DE VIGO

ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Memoria do Traballo de Fin de Grao que presenta

D. Daniel Camba Lamas

para a obtención do Título de Graduado en Enxeñaría Informática.

Herramienta gráfica para el diseño de componentes simples y estados para la definición estática de una interfaz gráfica de usuario.

Xuño, 2017

Traballo de Fin de Grao Nº:

Titor/a: Javier Rodeiro Iglesias
Área de coñecemento: Linguaxes e Sistemas Informáticos
Departamento: Informática

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

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.

Este proyecto da lugar a una herramienta que cubre esa fase del diseño de interfaces, pero siguiendo una definición formal de las mismas. Dicha definición está presente en el marco teórico de las tesis del tutor **Javier Rodeiro Iglesias [1]** donde se recoge la definición de *componente simple* y la continuación de dicha tesis por **Susana Gómez Carnero [2]** donde entre otra serie de ampliaciones, se recoge la definición de *estado*.

En el marco teórico citado se define que los componentes más básicos de una interfaz pueden definirse mediante dibujo o mediante imágenes, por motivos de alcance, en este proyecto nos basaremos en su definición mediante imágenes rasterizadas (BMP, JPG, PNG), siendo PNG el formato preferido debido al soporte de transparencias.

De tal forma que podremos definir formalmente un *prototipado falso* en base a *estados* estáticos y ordenados que contienen *componentes simples*.

Para obtener finalmente un archivo XML, donde estará definida la interfaz de manera jerárquica entre estados y componentes.

Y que posteriormente dicho archivo pueda ser cargado para realizar modificaciones en el prototipados o simplemente visualizarlo.

8. OBJETIVOS

- **OBJ-001:** Construir una aplicación gráfica de escritorio que resulte intuitiva respecto a otras aplicaciones de tratamiento de imágenes.

- **OBJ-002:** Gestión de las propiedades de los componentes simples, basados en imágenes:
  - Posición
  - Tamaño
  - Nivel de profundidad
  - Si es visible en la escena
  - Si está activo en la escena

- **OBJ-003:** Gestión de dichas propiedades en lotes (*sobre varios componentes simultáneamente*).

- **OBJ-004:** Gestión de estados, formados por componentes simples y sus propiedades:
  - Orden.

- **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 serán consideradas 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 falso* en un archivo XML donde de manera jerárquica y con etiquetas específicas, en base a los *componentes simples* y los *estados* que los contienen.
    - Cargar un archivo XML de forma que recupere el *prototipado falso* exactamente donde lo dejamos y poder así hacer modificaciones en los *estados* o *componentes simples* 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

Para alcanzar los objetivos de este proyecto, se ha utilizado una metodología iterativa e incremental dividiendo el proyecto en cinco bloques, siendo cada bloque una iteración. Esos cinco bloques van precedidos de una investigación a fin de conocer las mejores herramientas para el desarrollo eficiente y escalable de la aplicación.

Se ha utilizado el framework Qt para que la interfaz gráfica resultase multi-plataforma a nivel de ejecución y de interacción ya que elementos de interacción como los atajos de teclado, son traducidos a la plataforma que ejecuta la aplicación; también la librería i18n para que resultase multi-idioma.

Dado que este proyecto parte de un marco teórico ya desarrollado, y la especificación de dicho marco teórico está definida como un DTD de XML, la persistencia es hecha en dicho lenguaje y se utiliza lxml para cargar y guardar proyectos desde la aplicación.

El uso de Python ha sido únicamente por afinidad personal.

La solución propuesta es una aplicación con atención en la experiencia de usuario de manera que en la medida de lo posible el manejo resultará intuitivo a aquellos acostumbrados a otras herramientas con manejo de imágenes, ofreciendo los atajos de teclado usuales, desplazamiento de componentes mediante las flechas de dirección, siendo este desplazamiento de mayor amplitud si se pulsa la tecla Mayus (*Shift*), etc.

Teniendo en cuenta que es una aplicación gráfica, y que interactuamos directamente con la información, era necesario ofrecer al usuario un acceso al histórico de las acciones que realiza con los componentes que está utilizando para definir la escena, por ello con Ctrl+Z y Ctrl+Y (*o Ctrl+Shift+Z, dependiendo de la plataforma*) gestiona las entradas

del historial de acciones, desplazándote a las entradas previas con Ctrl+Z y pudiendo volver de dichas entradas con Ctrl+Y.

Cada estado que se crea, genera una entrada en la lista de estados, presente en la parte superior de la ventana de la aplicación y es identificada por la posición que ocupa (*esto es visible en el cabecero de la lista, donde también se indica con * el estado seleccionado en ese momento para edición*) y una miniatura que se renderiza en tiempo real en base a los cambios en la escena del estado.

Con toda la aplicación tiene un bajo consumo de recursos, el consumo de RAM inicial es de 60mb que aumentarán muy lentamente a medida que aumente el número de estados definidos, los componentes simples que estos contengan y las acciones realizadas, que se guardan para cada estado de manera independiente. Sin embargo, el coste en CPU de las operaciones que se permiten realizar es levemente superior a la de Photoshop.

Volviendo a la experiencia de usuario, características como: Que el zoom siga al cursor, el zoom sobre el área de trabajo, poder ocultar la totalidad de las barras de menú, tareas y listas, poder centrar automáticamente un componente en el área de trabajo, desplazar el área de trabajo, información en la barra de estado, etc. Son un valor añadido que seguro que agrada al usuario.

10. PLANIFICACIÓN Y SEGUIMIENTO

Gant y corregir tablas

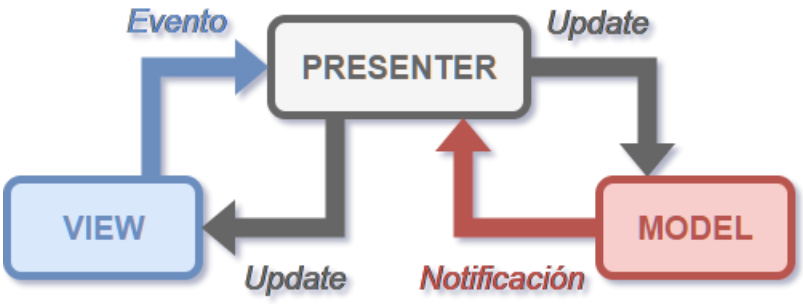
11. ARQUITECTURA

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 física 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

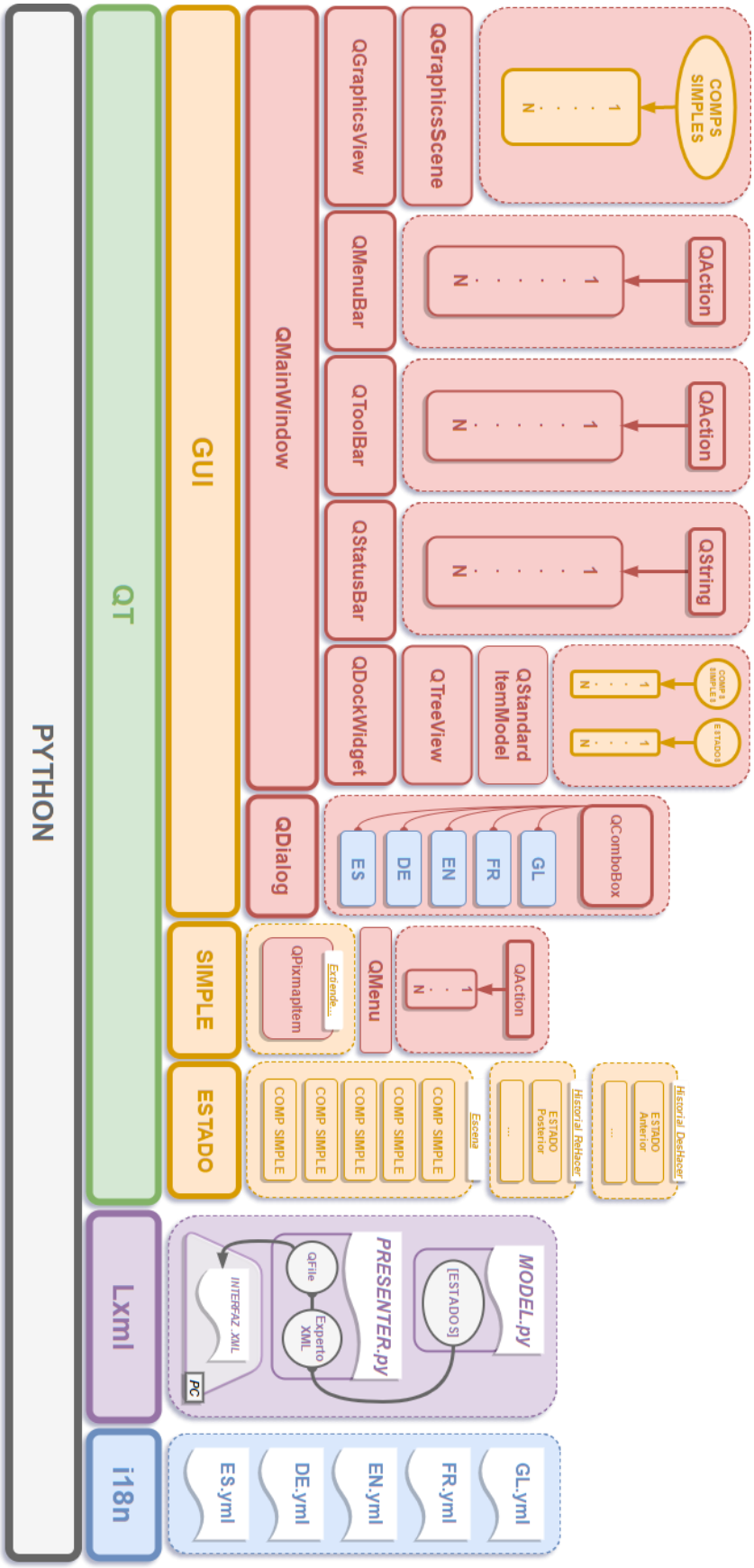


Figura 5.3.1. Diagrama de componentes, arquitectura de software.

En el diagrama podemos ver como el pilar de la aplicación es el lenguaje Python que a mayores de la librería estándar utiliza tres librerías externas, **i18n** para el soporte

multilenguaje, **lxml** para tareas de persistencia y **Qt** para la generación de interfaz y manejo de elementos gráficos. En detalle, los elementos de la librería Qt son utilizados para los siguientes fines:

GUI es un singleton con funciones que definen la creación y propiedades de los componentes Qt de los que se hacen uso para generar la interfaz y el área de trabajo.

QMainWindow da lugar a la ventana principal, donde se renderizará tanto la interfaz como el área de trabajo.

QGraphicsView y ***QGraphicsScene***, definen el área de trabajo, soportando la primera eventos de zoom y siendo la segunda el contenedor de los componentes simples con los que trabajaremos.

QMenuBar, ***QToolBar***, ***QStatusBar*** otorgan la estructura típica de cualquier interfaz de escritorio, representando (por orden de enumeración) una barra superior con los menús 'Archivo, Editar, Vista, Ayuda', una barra de herramientas para un rápido acceso a las acciones y una barra inferior para mostrar mensajes de ayuda o información de la interfaz.

QDockWidget define un panel que puede ser anclado en algún punto de la ventana principal.

QTreeView permite representar una información en forma de tablas o listas, según sea conveniente.

QDialog se encarga del menú de selección del idioma, antes de la carga de la interfaz de la aplicación.

SIMPLE es uno de los dos elementos principales que maneja la aplicación y es definido a partir de un componente Qt llamado ***QGraphicsPixmapItem***.

ESTADO es el otro elemento principal que maneja la aplicación, en su interior contiene la definición de la escena, la cual es un conjunto de componentes simples, y dos listas para almacenar estados anteriores y posteriores.

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 el soporte multi-idioma:

- **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

- **RF-001:** Definir un prototipado nuevo.
- **RF-002:** Guardar estado del prototipado en un fichero XML.
- **RF-003:** Carga el estado del prototipado a partir de un fichero XML con su definición.
- **RF-004:** Crear componentes simples.
- **RF-005:** Borrar componentes simples.
- **RF-006:** Dar nombre a un componente simple.
- **RF-007:** Duplicar un componente en el estado actual.
- **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 (W y H).
- **RF-010:** El software debe poder acceder al sistema de ficheros del usuario para cargar las imágenes.
- **RF-011:** Crear estado.
- **RF-012:** Eliminar estado.
- **RF-013:** Clonar estado.
- **RF-014:** Reordenar estados.
- **RF-015** Agregar componentes simples a un estado.
- **RF-016:** Convertir objetos de Python a XML, para guardar proyecto.
- **RF-017:** Convertir XML a objetos Python, para cargar proyecto.
- **RF-018:** Selección de idioma.
- **RF-019:** Acceso a historial de acciones sobre componentes simples.

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.
- **RNF-007:** Escalar el área de trabajo. (Manejo del zoom)
- **RNF-008:** Desplazar área de trabajo.
- **RNF-009:** Reseteo de escala del área de trabajo.
- **RNF-010:** Reseteo de la posición del área de trabajo.

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

14.1 Diseño estático

14.2 Diseño dinámico

15. GESTIÓN DE DATOS E 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 (*estados y componentes simples*) 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, el *modelo* alberga una cola de *estados*, y estos una cola de *componentes simples*; de manera que la aplicación renderizará en el área de trabajo los *componentes simples* del *estado* seleccionado en la cola del modelo.

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 *estado* en este proyecto, la relación entre *estados* y *componentes simples* es jerárquica, por lo que en el momento de guardar el prototipado que se esté definiendo en un momento dado, será suficiente comunicarle al patrón experto de XML, la cola de estados.

De forma que el experto (***Parser*** *en la implementación*) iterará cada *estado*, creando, con las etiquetas específicas, un nodo XML con la definición de cada uno, por lo que a su vez creará nodos con la definición de cada *componente simple* relativo al *estado*. Creando un archivo XML con la definición del prototipado y una carpeta con nombre <nombre-fichero-guardado>_imgs, donde se guardarán las imágenes, siendo necesaria dicha carpeta para la carga del prototipado definido en el XML.

Para realizar la carga de un prototipado, leeremos un fichero XML realizando la validación contra el DTD del marco teórico y construiremos los objetos pertinentes para manejar la definición del prototipado.

Los ficheros XML son almacenados con la codificación ISO-8859-1 para permitir el guardado y carga de imágenes y proyectos cuyos nombres contienen acentos o diéresis.

16. PRUEBAS

16.1 Validación del DTD

16.2 Test unitario de acciones

Creando una función que envie signals (pero deben de estar ya cargados los componentes… :/)

17. MANUAL DE USUARIO

17.1 Manual de instalación

17.1.1 Windows

17.1.2 Linux

17.1.3 Mac

17.2 Requisitos mínimos

17.3 Manual de Utilización

18. PRINCIPALES APORTACIONES

Desarrollar algo más 18.1 y 18.2 ¿??

18.1 Manejo de componentes simples

La aplicación es capaz realizar las siguientes acciones en relación a componentes simples: crear, eliminar, clonar, ver detalles, cambiar el nombre, modificar profundidad, modificar si el componente está activo, modificar si el componente es visible, centrar respecto a la escena, guardar historial de acciones aplicadas. Además dichas operaciones pueden realizarse en lote, sobre varios componentes a la vez.

Se relaciona con: OBJ-002, OBJ-003, OBJ-005, OBJ-006, OBJ-009

18.2 Manejo de estados estáticos

La aplicación es capaz de crear, eliminar, clonar y reordenar estados, por supuesto es capaz de agregar estados a dichos estados y lo más importante, es capaz de guardar dicha información en un fichero y volver a componerlos a partir de dicho fichero.

Se relaciona con: OBJ-001, OBJ-004, OBJ-007

18.3 Manejo del histórico de acciones por cada estado

Cada acción de cambio recibida por un componente simple, produce que se guarde el estado en el que el *estado* seleccionado se encontraba antes de que se realice la acción.

De esta manera uno puede recuperarlo utilizando Ctrl+Z o volver desde uno anterior al actual con Ctrl+Y siempre que no se haya realizado ningún cambio.

Dado que el desplazamiento de un componente de la posición 0, a la posición 100, son en verdad 100 cambios, pasando por todo el rango de 0 a 100; se ha decidido no guardar los estados intermedios, para evitar un drenaje de RAM y consumo poco productivo de CPU.

Se relaciona con: OBJ-001, OBJ-009

18.4 Experiencia de usuario (*y soporte multi-idioma*)

Mediante i18n la aplicación está disponible en cinco idiomas: español, gallego, inglés, francés y alemán.

Todas las acciones pueden ser invocadas desde atajos de teclado o mediante clics de ratón. Dichos atajos, y dichas acciones se valen de estándares de otras aplicaciones más conocidas y usadas.

Se relaciona con: OBJ-001, OBJ-005, OBJ-008

18.5 Inclusión de Z en la especificación formal

En el proceso de implementación, partiendo del DTD definido en el marco teórico, se comprendió que, para el correcto comportamiento de los componentes, era necesario el manejo de Z de forma que se pudiera modificar qué componente estaba detrás y cual estaba delante. Por lo que, tras tratar el tema con el tutor, el manejo de Z se incluyó en la implementación y consecuentemente, como la capa de persistencia es un XML que se validad contra el DTD, se ha modificado dicha especificación agregando dicho nodo a la etiqueta Enumeración.

Se relaciona con: OBJ-002, OBJ-003, OBJ-005, OBJ-006, OBJ-007, OBJ-009

19. CONCLUSIONES

19.1 Técnicas

La presente aplicación satisface la totalidad de los objetivos y añade factores de calidad en el marco de la experiencia de usuario. Si bien este proyecto es sólo una aproximación a la definición formal de interfaces que se recoge en el marco teórico citado en la introducción. Se han asentado unas buenas bases sobre las que construir una aplicación mayor que cubra la totalidad de la especificación.

19.2 Personales

En lo personal, este proyecto me ha hecho crecer y mucho, mi objetivo personal es profesionalizarme en el ámbito de la informática gráfica y con este primer acercamiento, no podría estar más satisfecho.

Ha habido momentos duros, interacciones que cuando usas la aplicación parecen extremadamente obvias, pero encierran muchas horas de investigación, pero se veían enormemente recompensadas, cada vez que un pixel de la pantalla hace lo que uno quería.

De hecho en base a la poca profundidad de los cursos sobre Qt existentes, con lo aprendido en el desarrollo de este TFG, bajo recomendación de Javi, he decidido realizar un curso de Qt para Udemy explicando la transmisión de datos entre clases usando señales en lugar de llamadas para tener una correcta arquitectura MVP, el manejo de QGraphicsPixmapItem (*componentes simples*), el manejo de Estados y como renderizar de manera eficiente las miniaturas y los componentes durante las operaciones básicas sobre éstos.

20. VIAS FUTURAS DE TRABAJO

20.1 ¿Dónde estamos?

La aplicación actual cubre un pequeño porcentaje de todo lo que abarca la definición formal de una interfaz, manejamos Estados y Componentes simples, los cuales son contenidos por los primeros, a mayores se ha refinado la especificación con la adición de la posición Z como nodo hijo en la etiqueta Enumeración. Eso a nivel de definición, a mayores la aplicación ya cuenta con un manejo de histórico, visualización en miniaturas, y opciones sobre la interfaz que mejoran la experiencia de usuario.

20.2 ¿Dónde queremos llegar?

El paso inmediatamente siguiente, sería a la aplicación actual, añadirle soporta para pincel; de forma que el usuario pueda crear líneas y determinadas figuras geométricas cuya definición está contemplada en la especificación, de forma que dichas figuras pudieran ser manejas por la aplicación del mismo modo que maneja ahora los componentes simples creados a partir de imágenes.

21. REFERENCIAS

22. APARTADOS ADICIONALES