Juced

JUCE Interface Designer

Herramienta de desarrollo de Interfaces Gráficas personalizables

Miembros del equipo desarrollador:
Raimon Beltran Ràfols
Daniel Guija Alcaraz
Cristian Lopez Garcia

Projectes de les Tecnologies de la Informació - PTIN
UPC - EPSEVG
2013

Bienvenido al proyecto
JUCED Juce Interface Designer

Esta herramienta de desarrollo rápido de aplicaciones está diseñada para permitir a los usuarios dibujar una Interfaz Gráfica de Usuario mediante la biblioteca JUCE con unos pocos clics de ratón.

Sumario

En los inicios del proyecto JUCED se pensó en cómo poder proponer algo nuevo y al mismo tiempo útil, teniendo en cuenta el estado actual del software de desarrollo de interfaces gráficas, para poder dar un nuevo giro de tuerca al panorama centrado en el desarrollo de interfaces, un sector en el que la relación entre el usuario y el software es cada vez más exigente pero al mismo tiempo más cercana. A partir de esa premisa se han ido creando los principales valores en este proyecto para cubrir aquellas necesidades que surgen en el mercado de las nuevas tecnologías. A medida que la tecnología avanza, con nuevas formas de interactuar con nuevos terminales y dispositivos, y en definitiva tanto con software como hardware innovador, es necesario buscar nuevas sendas que también innoven y permitan una mayor interacción, comodidad y personalización, para una cada vez más amplia gama de usuarios, proporcionando una mayor agilización en el proceso.

El proyecto de JUCED propone el desarrollo de las herramientas que permitan a los desarrolladores y diseñadores ser capaces de crear sus propias interfaces gráficas para distintos perfiles de usuarios y para diferentes tipos de ámbitos. Para ello se introducen las características necesarias para un desarrollo ágil, sencillo pero potente, siendo capaz de visualizar y testear en tiempo real sin mayores pasos intermedios, en una diversidad de opciones disponibles para el desarrollador en tareas de implementación y validación de las interfaces gráficas. Pero además de personalización, ofrecer versatilidad en cuanto al destino de estas herramientas, siendo útiles para distintos campos en los que una interfaz entre aplicación y usuario sea de total importancia.

El producto final JUCED en su conjunto es una herramienta actual, moderno, potente y capaz que ofrece beneficios tanto al desarrollador/diseñador como al cliente. Una de las primeras motivaciones es poner al alcance las herramientas necesarias que proporcionan dotar a cualquier aplicación de una interfaz capaz de conectar con el usuario de forma funcional, cómoda y amigable mientras que, además, se pueda conectar con la aplicación de una forma ajustada a cualquier preferencia, facilitando su uso de forma intuitiva y adaptada. JUCED ofrece un editor que no reinventa la rueda, sino que toma las virtudes de los editores de interfaces gráficas de usuario actuales y las lleva un paso más allá intentado eliminar las complejidades ofreciendo un aplicativo de código moderno y actual para diversos propósitos en el campo de las tecnologías de la información siendo simple, pero a la vez avanzado.

Índice

1. Introducción

El proyecto JUCED nace ante la necesidad de desarrollo de una herramienta que permita a los usuarios, desarrolladores e incluso diseñadores, la personalización de interfaces, para satisfacer las necesidades de un importante cliente del sector de las tecnologías de la información. Nosotros, la empresa PTIN asociada a la universidad UPC - EPSEVG, concretamente al departamento Proyectos de las Tecnologías de la Información, se encargará del desarrollo de una herramienta que sea capaz de adaptar una GUI o Interfaz gráfica de usuario al gusto y deseo de un amplio abanico de usuarios de la misma herramienta, con el objetivo principal de satisfacer sus necesidades según la problemática o uso que el usuario vaya a poner en práctica.

Partiendo de la premisa anteriormente citada, se lleva a cabo el inicio del proyecto JUCED, una herramienta constructora de interfaces gráficas para el cliente y/o usuario final. Es necesario mencionar la naturaleza de la herramienta, siendo ésta un medio de desarrollo y programación el cual simplifica y asiste en la creación de interfaces gráficas del usuario que permite a un desarrollador o diseñador la visualización y automatización del código fuente partiendo de cada parámetro, correspondiente por ejemplo a widgets o módulos.

La herramienta JUCED pretende facilitar la creación, y aún más la personalización, de interfaces gráficas que servirán para distintos propósitos designados por el desarrollador, y más allá de ello ofrecer una versatilidad para la infinidad de posibilidades que ofrece hoy en día el mundo del software. En vez de centrarse en un único campo, como puede ser el de Networking o Management, la herramienta puede ser flexible para poder adaptarse y ofrecer soluciones a múltiples campos de las Tecnologías de la Información. De esta manera, puede decirse que el resultado del proyecto supone un medio para un fin.

El proyecto para el desarrollo de la herramienta cuenta con un periodo de tiempo limitado a 12 semanas, y es por ello que el desarrollo se destina a ciertas funcionalidades y características dentro del mismo, siendo el resultado final del proyecto de naturaleza abierta y extensible.

A continuación, tras esta introducción, se da paso al resto de la documentación que conforman la memoria del proyecto en sí, donde todos los aspectos aquí documentados pueden estar sujetos a cambios y no son representativos del producto final.

2. El equipo

El equipo encargado del desarrollo del proyecto JUCED es un pequeño grupo de personas que conforman la empresa llamada PTIN. Somos una pequeña empresa emprendedora asociada a la universidad UPC - EPSEVG, colaborando con el departamento de Proyectos de las Tecnologías de la Información. El equipo está integrado principalmente por tres miembros, fundadores de la empresa, quienes se encargan de las principales tareas de todo proyecto: planificación, especificación, diseño, implementación y validación. Además, el equipo principal suele incluir la colaboración de otras compañías, por lo que entre nuestras filas pueden encontrarse otros miembros de colaboración, dependiendo de las necesidades y exigencias de los proyectos con los que podemos trabajar.

Nuestra empresa PTIN puede decirse que es una empresa joven, que ha trabajado con anterioridad en proyectos de pequeño tamaño para clientes PYME dentro del campo de las tecnologías de la información. Nuestros proyectos anteriores se remontan desde prototipos y diseños sin llegar a ser un producto final, hasta el desarrollo de pequeñas aplicaciones para sistemas de sobremesa.

El proyecto JUCED es un proyecto de mayor envergadura que los anteriores desarrollados por la empresa, destinado a cumplir las expectativas de un importante cliente. El proyecto se planifica para una duración de 12 semanas con el motivo de obtener un producto inicial terminado, candidato para ser liberado al público. Dada la naturaleza abierta y extensible del proyecto y cumpliendo con las necesidades del cliente, el proyecto JUCED aspira a crecer y ampliar sus objetivos una vez la primera versión de la herramienta se haya completado, y por consiguiente, lanzada al público por nuestro cliente, encargado de la distribución.

3. La Propuesta de Proyecto

En esta primera sección de la documentación del proyecto se muestran los objetivos fijados para el cumplimiento del producto, un acercamiento y visión del sistema, como también de su arquitectura, y el funcionamiento del conjunto. Se detallan los actores y componentes que conforman el entorno, así como las interfaces que lo conectan y permiten interactuar los actores con los componentes.

3.1 Objetivos y Requerimientos:

En este apartado se muestran los principales objetivos que marcan el desarrollo de la herramienta. Los objetivos mostrados aquí se han fijado partiendo de las limitaciones que muestra el proyecto, por lo que otros objetivos se han descartado, pero no implican que sean los únicos ni tampoco excluyentes. Además, cabe destacar que todos los objetivos han sido fijados para su cumplimiento de forma progresiva y pueden variar de un modo u otro en el desarrollo final. Para conseguir un mayor seguimiento de progresos, y simplificar la distribución de tareas y de la fuerza de trabajo, se han tomado una variedad de objetivos concretos, con alta granularidad y distinción entre ellos, englobando el conjunto de metas necesarias para la construcción del prototipo final.
Los objetivos son independientes de cualquier tecnología, y su enumeración no indica ningún tipo de orden o prioridad; las prioridades se detallan en el plan de trabajo.

1. Edición de GUIs: Otorgar a los desarrolladores las herramientas que permiten crear interfaces gráficas de manera personalizada, eliminando las complejidades que pueden suponer los desarrollos de interfaces de uso final, simplificando el proceso y facilitando la interacción con el usuario a la vez que ofrecer opciones más avanzadas.

2. Versátil: Capacidad de permitir a los desarrolladores con diversas opciones de personalización la implementación de interfaces, de forma multi-propósito, a través del desarrollo desde cero o de plantillas, o de forma más específica y orientada a sectores más concretos.

3. Modular y extensible: Disponer de un entorno de trabajo sobre la aplicación que permita ofrecer la facilidad de interacción con el desarrollador a través de conjuntos o módulos, y la capacidad de extensión con la cual se puedan incluir nuevas características al conjunto en un futuro si es deseado.

4. Independiente: La herramienta debe de ser capaz de funcionar de forma independiente a otro tipo de software aplicación y de sistema operativo en el cual se vaya a utilizar.

5. Personalizable: La herramienta debe tener la capacidad de adaptarse a la visión del desarrollador y a las necesidades de cada proyecto, ofreciendo sistemas de modificación de apariencia de interfaz, módulos, widgets y entornos.

6. Importación y exportación: La herramienta debe permitir a los desarrolladores poder extraer el desarrollo de su interfaz a ciertos formatos que luego puedan ser utilizados con terceras aplicaciones para los proyecto designadas, así mismo como importar y adaptar interfaces previamente desarrolladas.

7. Caja de herramientas: Crear un entorno o marco de trabajo a modo de editor en el que se integren conjuntos de herramientas, módulos y opciones para la creación de todo tipo de objetos propios de cualquier interfaz, por ejemplo: ventanas, menús, botones, pestañas, etc. y la edición de éstos pudiendo dibujar, o aplicar drag and drop e incluso modificar parámetros.

8. Las herramientas: Ofrecer la capacidad de introducir componentes en el entorno de trabajo, principalmente para cubrir cualquier necesidad de desarrollo, creando una estructura que formará la interfaz gráfica. Las herramientas ofrecen variedad de opciones y pueden ser muy diversas. Estas herramientas pueden ser las citadas anteriormente entre muchas otras, y forman parte de la caja de herramientas.

9. Edición de propiedades: Ofrecer el acceso necesario a los parámetros de cada módulo o componente para poder visualizar, editar o rectificar cualquier parámetro y atributo en tiempo real.

3.2 Visión del Sistema y Arquitectura

En este apartado se muestra una visión global del sistema y la estructura que conforma el entorno.

El proyecto JUCED como producto final destinado a desarrolladores, diseñadores o usuarios más genéricos, se compone como un paquete independiente donde su uso empieza por la instalación del mismo producto dentro del sistema operativo del equipo, sin exigir ningún tipo de dependencias ni instalaciones adicionales.

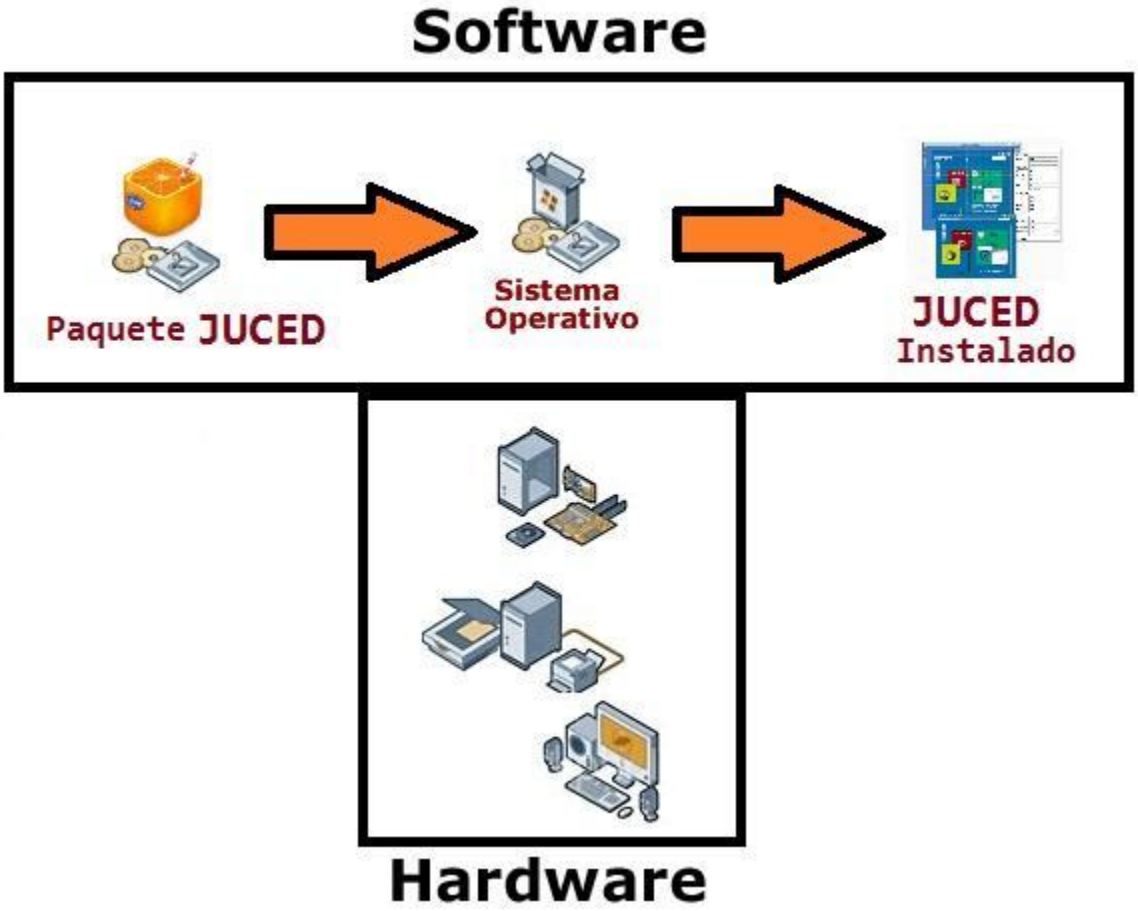


Imagen que representa el producto en el marco físico de uso.

La arquitectura del sistema con un enfoque global y en primera instancia el sistema (el conjunto de elementos que forman la totalidad de la herramienta) está compuesto por la herramienta en sí, además de la entrada y la salida de datos, que llamaremos Input y Output y que son los componentes que dan funcionalidad al sistema.

El Input serán todos aquellos datos que el usuario de la herramienta introduzca , ya sea código fuente, templates, importaciones, otro tipo de datos e incluso interacción.
El Output serán todos aquellos datos que el usuario reciba o extraiga de la herramienta, que mediante el entorno de usuario permita obtener datos guardados, exportados, así como también interfaces editadas con la herramienta, donde el usuario puede recoger en el formato deseado el producto de su desarrollo (por ejemplo en XML).

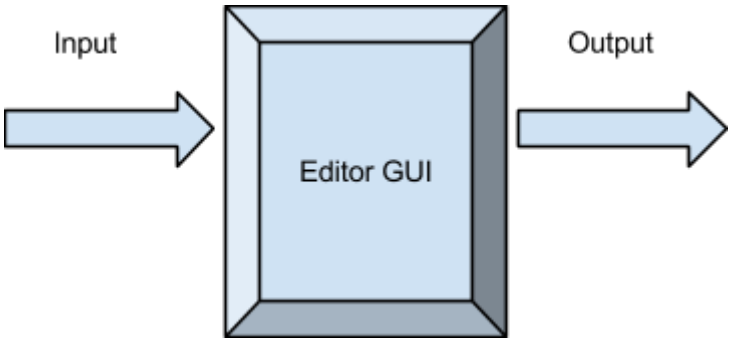


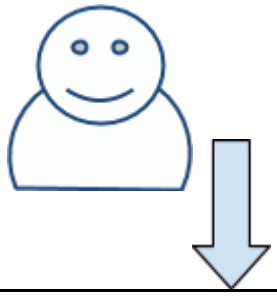
Imagen esquemática de interacción y flujo de datos de la arquitectura.

Dada la naturaleza abierta de la herramienta, y que uno de sus objetivos principales es la modularidad y extensibilidad, otros ejemplo de Input y Output son la capacidad de introducir nuevas funcionalidades, características, módulos, como también la exportación de éstos si a través de la herramienta se efectúa algún tipo de modificación, como es la personalización en sí.

A continuación se muestra un enfoque más concreto de la arquitectura del sistema con el objetivo de estructurar el entorno que presenta la herramienta.

El modelo de capas de la arquitectura muestra las distintas capas en orden jerárquico que forma el conjunto de la herramienta. Como se muestra a continuación las capas y el orden son los siguientes: Interfaz, Módulos, Aplicación, Sistema Operativo, Hardware. Cada capa actúa a la vez con la inferior y la superior, excepto la última (Hardware) y la primera (Interfaz). La capa de Interfaz es el enlace del usuario (o desarrollador) que conecta la interacción de éste con el resto del sistema, mientras que la capa de Hardware es la base donde se asienta el sistema.

A continuación una imágen ilustrativa de la arquitectura estructurada en capas y como el usuario interactúa con el sistema:



Interfaz
Módulos
Aplicación
Sistema Operativo
Hardware

Modelo de capas de la arquitectura de la herramienta

Como se muestra en la figura superior, la capa interfaz es la encargada de conectar la herramienta con el usuario, y hace de puente a la vez entre el usuario y la disposición de módulos de la aplicación, que se encargarán de integrar conjuntos de herramientas y utilidades que el usuario podrá utilizar para diseñar y construir sus propios proyectos.

3.3 Actores y componentes

La arquitectura del entorno se divide principalmente en actores y componentes. Los actores se encargan de ofrecer uso / interacción entre otros actores y componentes, y son críticos en la arquitectura del sistema. Los componentes (no confundir con los módulos o componentes propios de la aplicación) son objetos que actúan como nexo entre los distintos actores que conforman el diseño.
Los principales actores y componentes que conforman el entorno de la herramienta JUCED son los siguientes:

Actores	Componentes
Usuario/Desarrollador	Interfaz
Datos Importados/Exportados	Módulos / Widgets
Templates	Extensiones
Aplicación	Propiedades / Atributos
Sistema Operativo	
Hardware	

Tabla de distribución de actores y componentes según la arquitectura de la herramienta

3.4 Interfaces

En este apartado se muestran y se explican las interfaces que relacionan e interactúan entre los distintos actores y componentes que conforman el entorno de la herramienta:

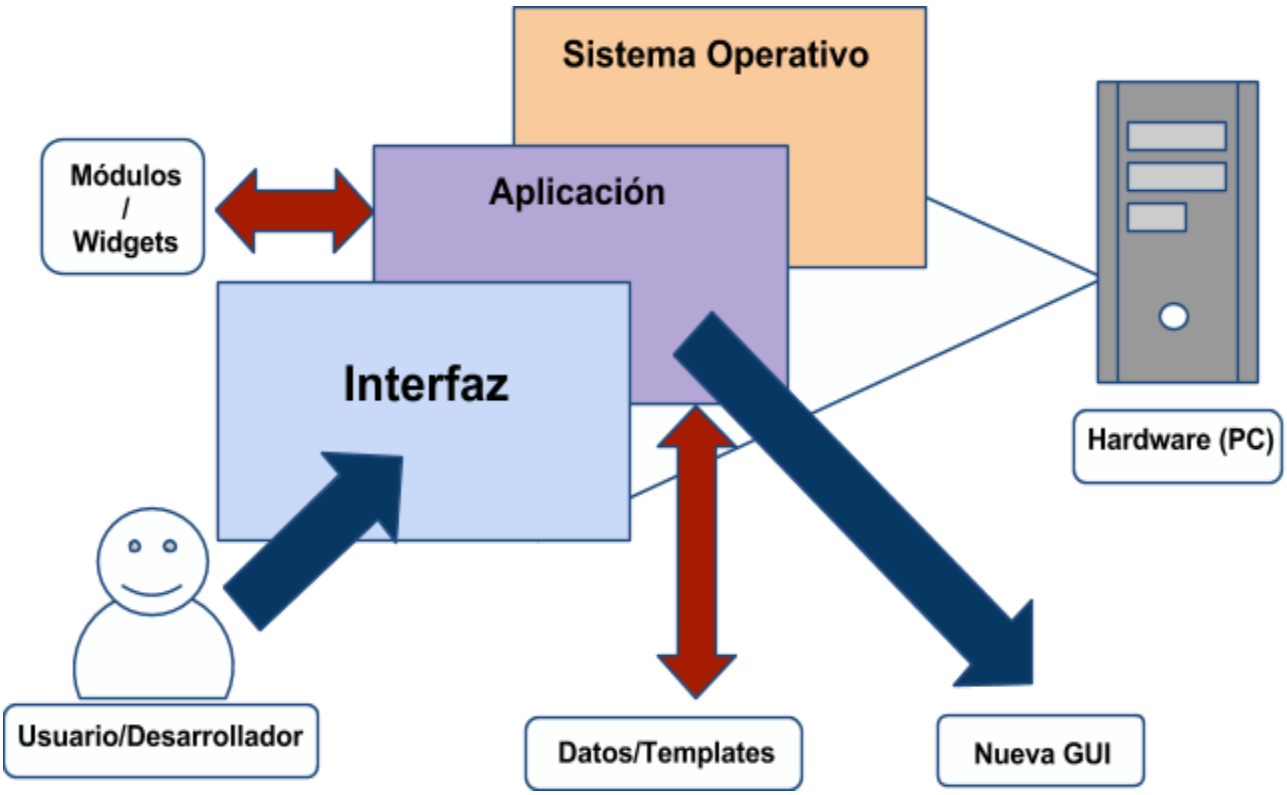


Figura que muestra las relaciones entre actores y componentes de la arquitectura de la herramienta

En la figura anterior podemos ver que existen tres flujos distintos que relacionan los distintos actores y componentes de la herramienta.

El flujo azul se centra principalmente en la interacción del usuario con la interfaz de la herramienta. El usuario se encarga de realizar las acciones necesarios (relación de Input del usuario sobre la interfaz) mediante las cuales, a través de la herramienta (la aplicación en sí) se obtiene lo que se llama una nueva GUI, que puede disponerse en distintos formatos como proyecto creado por el usuario, ya sea a través de exportación o guardado del trabajo realizado.

El flujo rojo es que relaciona la introducción y extracción de contenido de la aplicación, donde por ejemplo, a través de las opciones de extensión el usuario puede introducir

nuevas herramientas y utilidades contenidas en módulos o widgets, así como introducir o extraer plantillas y otros datos de la aplicación.

El modelo de capas anteriormente visto se ve reflejado sobre las capas de la interfaz, que opera sobre la aplicación y a la vez el conjunto de la herramienta funciona sobre el sistema operativo.

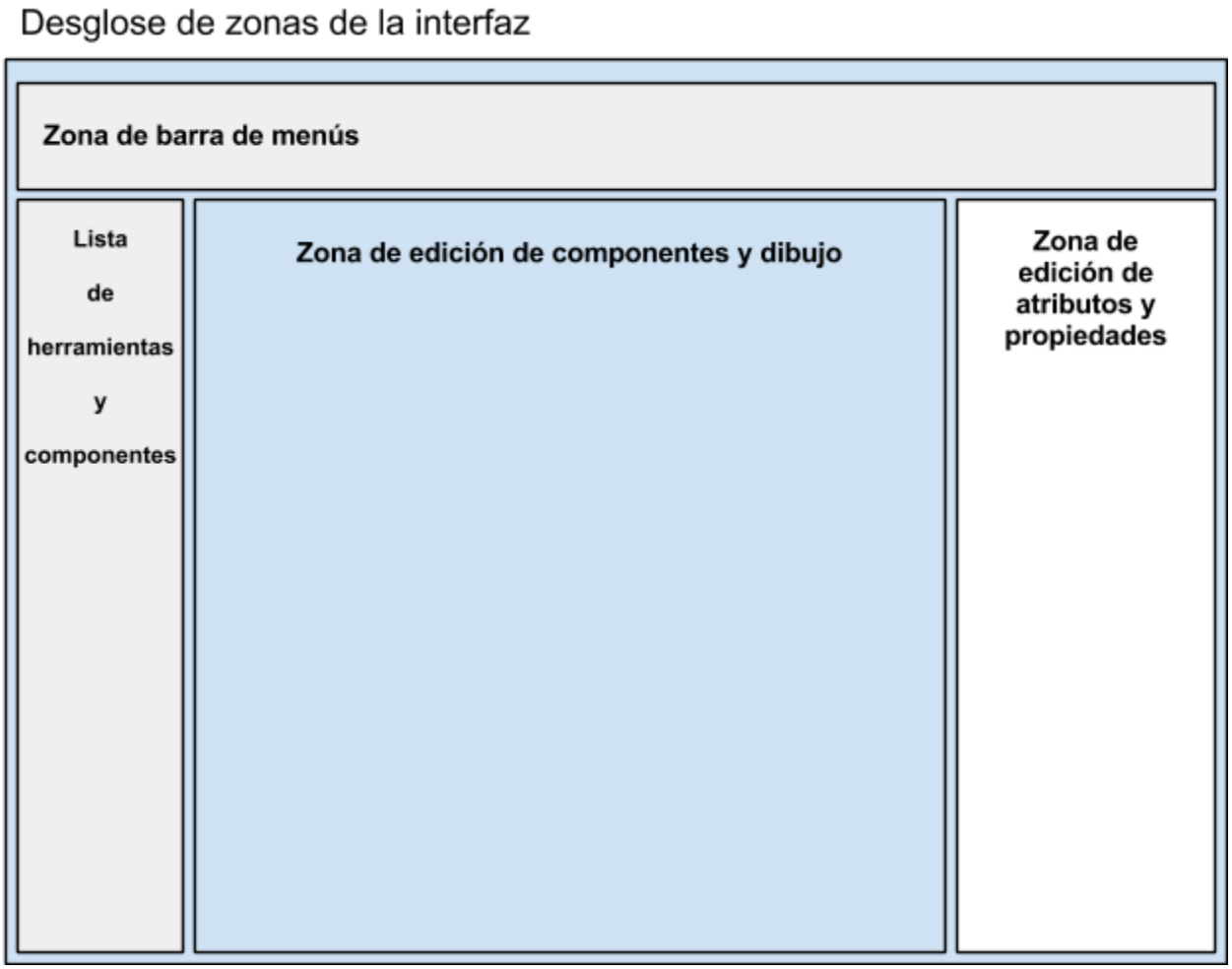


Figura que muestra una aproximación de las distintas zonas de interacción del usuario con la interfaz

3.5 ‘State of the Art’ y selección de tecnologías

Este apartado se centra en la discusión y selección de las tecnologías adecuadas para llevar a cabo el proyecto JUCED. Es en este apartado donde empieza a cobrar importancia el conjunto de tecnologías y el software necesario para poder diseñar e implementar el proyecto JUCED. Como el nombre de la sección indica, el ‘State of the Art’ del software de creación y diseños de interfaces gráficas es muy importante y supone uno de los puntos vitales para emprender el proyecto.
Principalmente se han considerado todas aquellas herramientas que a día de hoy ofrecen muchas posibilidades en el terreno de las interfaces gráficas, pero que además la cantidad de documentación disponible facilite el trabajo para aprovechar el tiempo disponible de desarrollo. Uno de los puntos claves que se tienen en cuenta es poder desarrollar el proyecto sin reinventar la rueda, es decir, ofrecer algo relativamente nuevo y moderno, sin caer en el error de hacer lo que ya está hecho. A continuación se muestra algunas de las tecnologías candidatas para el desarrollo de JUCED:

QT Creator:

Nos documentamos sobre Qt y alternativas debido a que Qt Designer puede funcionar para cualquier lenguaje de programación soportado por Qt.

Además, tuvimos en cuenta que Qt tiene restricciones en su licencia de uso, que por llevar a cabo determinadas tareas se debe pagar. Como por ejemplo, compilar las librerias de Qt dentro de un mismo programa para crear solo un archivo binario sin dependencias.

Uno de nuestros objetivos principales es que el proyecto JUCED pueda ser libre y extensible, sin dependencias de terceros. Por lo tanto, QT pasaba a segundo plano.

WxWidgets:

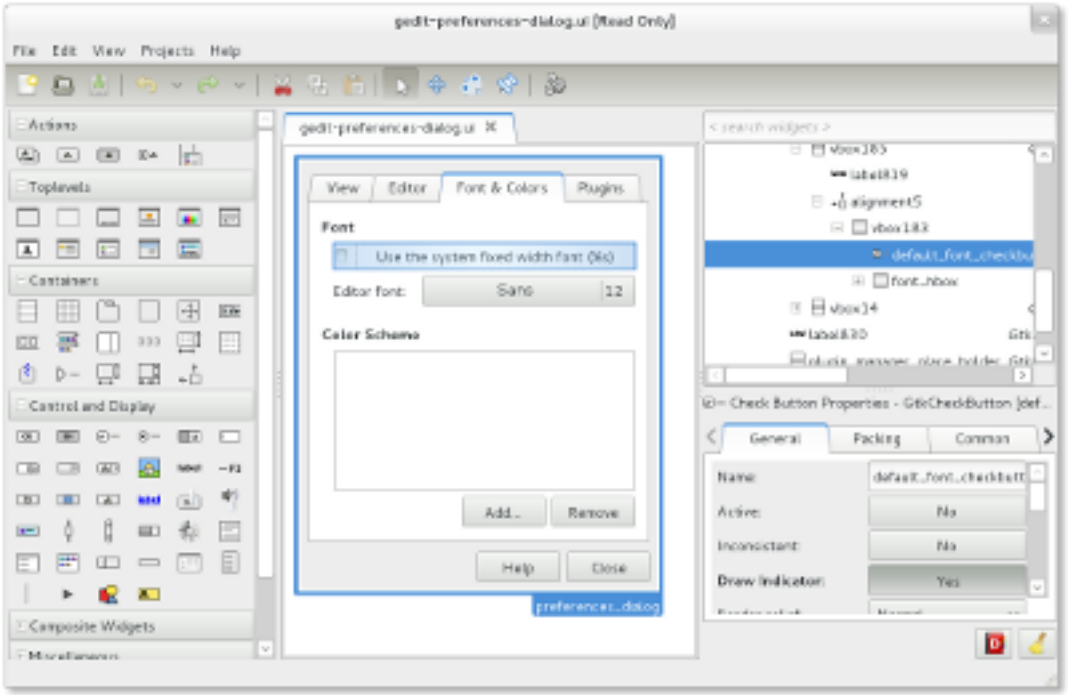
wxWidgets es una biblioteca de C++ que permite a los desarrolladores crear aplicaciones para Windows, OS X, Linux y UNIX en arquitecturas de 32-bit y 64-bit, así como varias plataformas móviles como Windows Mobile, iPhone y SDK incorporado GTK +. Tiene enlaces populares de idiomas para Python, Perl, Ruby y muchos otros idiomas. wxWidgets da a sus aplicaciones un aspecto natural ya que utiliza la API nativa de la plataforma en lugar de emular la interfaz gráfica de usuario. También es amplio y gratuito, de código abierto y maduro.

WxWidgets fue una gran candidata para ser la base de desarrollo de JUCED, a través de GTK+ nos ofrecía una buena opción pero fue descartada por los motivos mostrados más adelante.

GTK+:

Sobre wxWidgets había mucha opinión variada, por lo que nos centramos directamente en GTK+.

GTK+ dispone de una herramienta para diseñar interfaces llamada Glade:



Es Open source, pero solo permite diseñar los distintos tipos de ventanas y exportarlas a XML, luego cargarlas a un programa mediante código, llamando al GtkBuilder que construye una interfaz mediante XML. Esta aplicación nos permite añadir nuevos widgets, sin embargo Glade no dispone de proyectos ni edición de código, sólo crear nuevas ventanas y obtener su XML. Tampoco dispone de templates.

A partir de estas tres principales opciones aquí mostradas para el desarrollo, el equipo se planteó utilizar GTK+ sobre QT:

1. Programar nuestra propia herramienta Glade, pero suponía reinventar la rueda e iba contra uno de los principios del proyecto.

2. Hacer un fork del código fuente, creando una nueva versión donde desarrollar e integrar las opciones de proyectos y templates, de forma que permitiera crear proyectos para un lenguaje específico y que crease el código base pudiendo elegir un tema o plantilla específico o importar otro proyecto.

3. Crear nuestra aplicación, que ofrezca principalmente edición de código, creación de proyectos, etc. e integrar la "Glade UI Designer core library" para la edición de GUIs.

Finalmente, el equipo se dio cuenta que ninguna de las opciones encajaba con la motivación y los propósitos principales de proyecto. La tercera opción había sido la que más encajaba con las exigencias del proyecto JUCED, sin embargo la investigación y el estudio de tecnologías no se quedó ahí.

Al inicio del proyecto, cuando éste era agnóstico a las tecnologías que iban a ser empleadas, el proyecto utilizó el nombre de CUSTOM GUIs Designer, sin embargo su nombre acabó siendo JUCED. Esto se debe a que el equipo descubrió la que iba a ser la tecnología definitiva para el proyecto: se llama JUCE, las librerías que a continuación se detallan, y se debe precisamente a el nombre de dichas librerías, el nombre que da forma a este proyecto, JUCED.

JUCE (Jules' Utility Class Extensions):

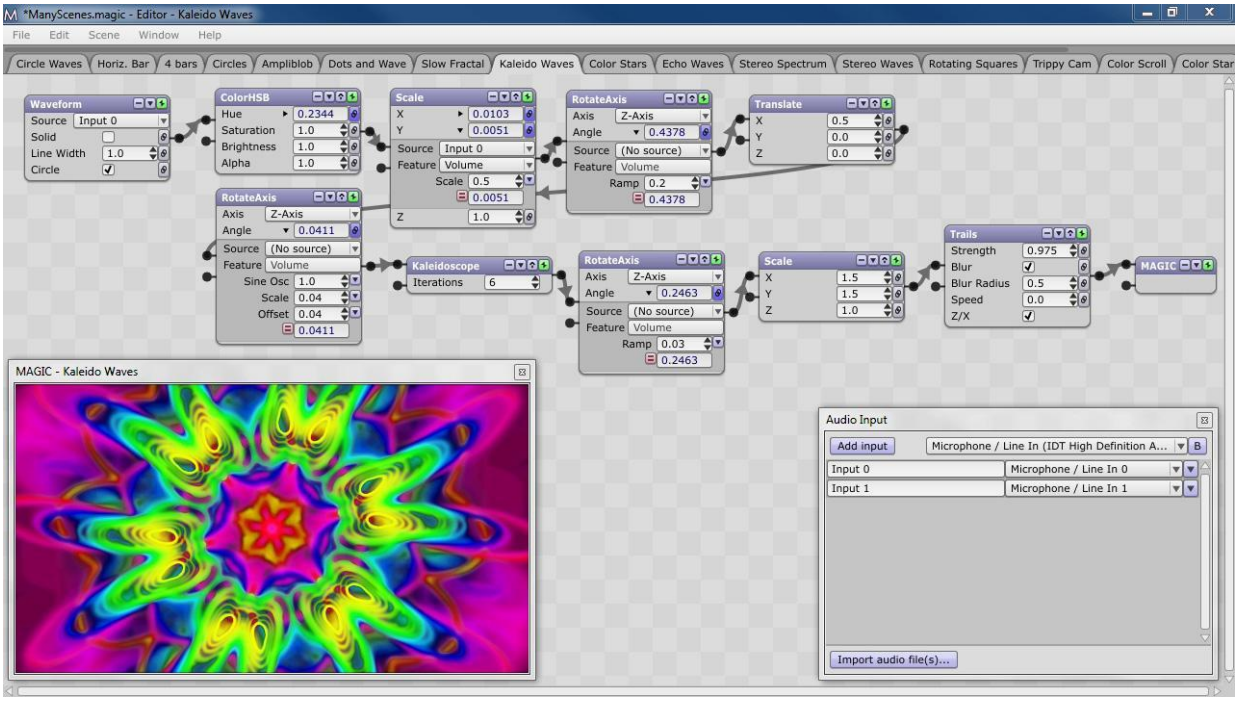
JUCE es otra librería gráfica realmente potente y con un gran diseño. Se trata de unas librerías de C++ que abarcan toda clase funcionalidades para el desarrollo de software multiplataforma.

Contiene casi todo lo que es probable que necesita para crear la mayoría de las aplicaciones, y es especialmente adecuado para la construcción de interfaces gráficas de usuario altamente personalizadas, y para el manejo de gráficos y sonido.

A continuación se muestran algunos ejemplos de productos del mercado actual que utilizan las librerías JUCE.

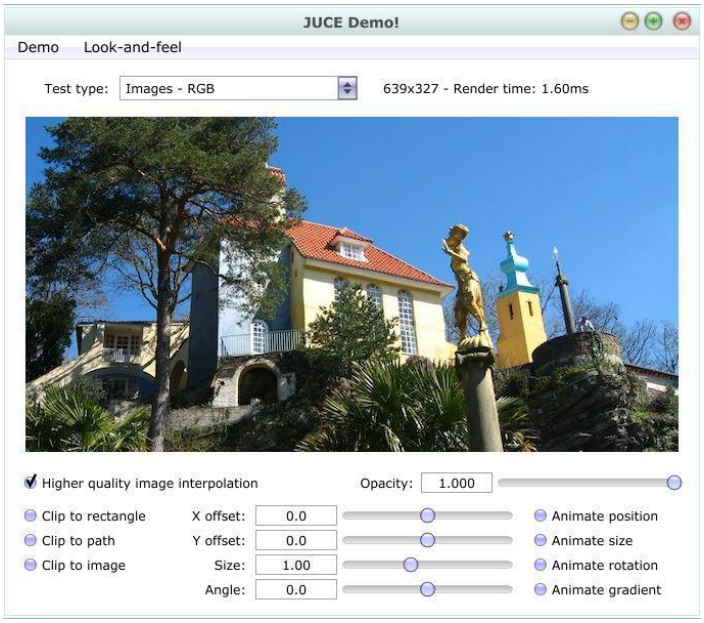
Ejemplo de aplicación comercial que usa JUCE:

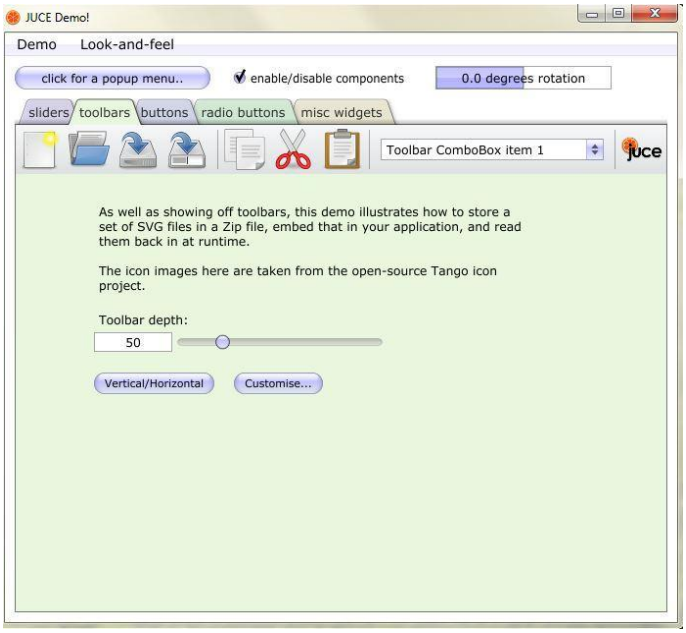
MAGIC: Multitrack Audio & Graphics Interactive Composer



Demo ejecutable mostrando las funcionalidades de JUCE

A continuación se muestran algunas capturas de la Demo de JUCE que contiene muchas de las funcionalidades de gran utilidad para la creación de interfaces gráficas, aunque el aspecto es minimalista, ya que para demostrar su potencial colorea los componentes a su gusto sin seguir un patrón de diseño. La demo puede ser compilada con Visual Studio 2010 SP1 (cualquier edición) mediante uno de los códigos fuente de ejemplo con los que viene JUCE.





Ambas capturas muestran distintas funciones que ofrece JUCE, tanto con apariencia nativa como sin ella.

4. Metodología y Plan de trabajo

En las siguientes secciones se tratan todos los temas relacionados con los métodos y el plan de trabajo para llevar a cabo el proyecto JUCED. En primera instancia se trata de preparar las bases en pequeños lotes de tareas que se van distribuyendo a lo largo de las 12 semanas del plazo de entrega. Para ello se prepara una primera aproximación que se divide en tres fases para el prototipo final, antes de pasar una última fase de validación.

4.1 Método

Prototipo: 3 Fases - El prototipo se divide en tres fases mostradas a continuación, donde las dos primeras corresponden al período de 12 semanas del proyecto, y la última fase corresponde a la fase de extensión del producto final.

LEYENDA:

Fase inicial: Objetivos que deberíamos cumplir en unas 2-5 semanas.
Fase avanzada: Objetivos a cumplir dentro del período de entrega.
Fase extendida: Objetivos que están fuera del alcance del proyecto debido al corto plazo de tiempo.

Visión general: Los principios

Nuestra primera meta debe ser el desarrollo de un conjunto de componentes que puedan ser añadidos a cualquier proyecto desarrollado en JUCE, particularmente Introjucer, una herramienta IDE para la creación y gestión de proyectos. No es posible desarrollar estos componentes dentro del IDE Introjucer ya que éste actualiza su código fuente con frecuencia y se tendría que estar siempre adaptando a las nuevas versiones. De modo, que se debe partir de un proyecto vacío con una ventana y su correspondiente contenedor de componentes. En este contenedor de componentes se inicializa nuestro editor y llama a sus mètodos para obtener los distintos componentes necesarios para el editor, como las toolbox o la ventana de propiedades de un componente, por ejemplo. A partir de éste punto se puede empezar a construir la base del proyecto, donde se podrá ir desarrollando cada una de las etapas mostradas a continuación, y que nos ayudará más adelante para definir los milestones y los deliverable del proyecto.

Estructura VALUETREE

Se llama ValueTree una estructura de árbol muy potente que puede ser usada para mantener datos de forma libre, y que puede manejar su propio comportamiento de deshacer y rehacer.

ValueTree contiene una lista de las propiedades con nombre como objetos variables, y también contiene un número de subárboles.

Crea objetos ValueTree en la pila, que son simplemente una referencia de peso ligero para un contenedor de datos compartidos. La creación de una copia de otro ValueTree simplemente crea una nueva referencia al mismo objeto subyacente, pero así mismo puede hacer una copia por separado, en el fondo de un árbol de forma explícita.

ValueTree puede llegar a ser mucho más complejo, pero pensar en ello y utilizarlo puede ser muy útil y a la vez ofrecer muchas ventajas para el diseño y la implementación del núcleo de la herramienta, porque a través de esta estructura se permite principalmente deshacer y rehacer cambios. Por lo tanto, una de las bases del proyecto JUCED es BigTree, una variante de los ValueTree.

A continuación se cita una primera aproximación de las principales características que requiere JUCED y que conforman las tareas que más adelante se distribuirán en la planificación.

Características del editor

1. **Toolbox**: Permite seleccionar una herramienta que corresponde a un componente para ser dibujado en el editor.

2. **Interacción con el editor**:
    a. Pinta el área que va a ocupar el nuevo componente mientras el usuario lo dibuja en el editor. Crea un nuevo componente dentro del componente en el que empezó el drag&drop.

    b. Selecciona un componente en el editor para poder editar sus propiedades, redimensionar y desplazar.

3. **Propiedades**: Permite modificar los atributos de un componente.

4. Componentes capaces de ser dibujados por el editor:
    a. **Window**
    b. **Label**
    c. **Textbox**
    d. **Button**
    e. **Slider**
    f. **Listbox**
    g. **Combobox**
    h. **ImageButton**
    i. **Directshow**
    j. **ImageComponent**
    k. **ImageButton**
    l. **Tabs**
    m. **TreeView**
    n. **Viewports**
    o. **Tables**

5. **Historial de cambios**: Permite rehacer/deshacer mediante el componente BigTree.

6. **Exportar**: Guarda la estructura de componentes y sus atributos en un archivo XML.

7. **Importar**: Carga la estructura de componentes y sus atributos de un archivo XML.

8. **Generar código**: genera los archivos de código fuente necesarios para poder compilar la interfaz.

9. Formas de editar el código por parte del usuario:
    a. **Gestionar eventos**: Proporciona funciones en archivos de código fuente separados del código generado para el tratamiento de eventos.
    b. **Edición directa de código**: Permite al usuario añadir líneas de código dentro del código generado, manteniendo un constante registro de cambios.

10. **Componentes externos**: Permite al usuario añadir librerías externas (*.dll) al componente editor.

Características de la aplicación

Hay otras características que no forman parte del componente editor sino de la aplicación que alberga el editor, y que nos ayudan a detallar los milestones de la parte de la aplicación:

1. **Plantillas**: Permite seleccionar un modelo de aplicación específico al crear el proyecto.

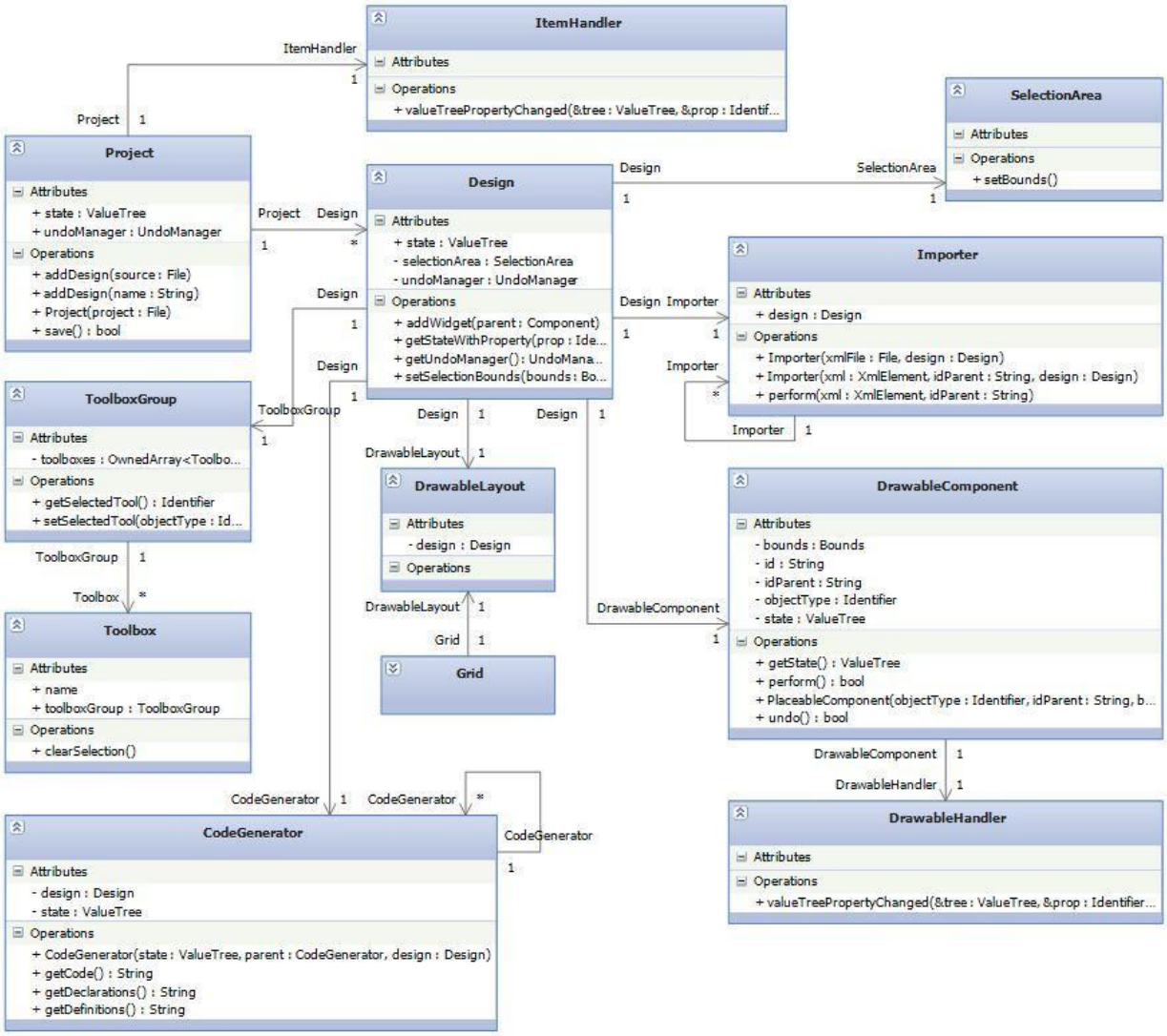
2. **Interfaz para interactuar con el editor**: Proporciona componentes que permiten importar, exportar, gestionar los atributos del editor (p.e. ancho de la rejilla de fondo).

3. **Creación de componentes**: Permite crear un proyecto componente para ser compilado como una librería externa (*.dll).

4. **Añadir componentes personalizados**: Permite al usuario incorporar librerías externas de componentes en el editor, así como mostrar su herramienta para dibujarlos en una de las toolbars.

Principio de diseño de Clases

Bajo estas líneas se muestra un primer diagrama de clases el cual representa el núcleo (o base, como más adelante llamaremos) de la aplicación utilizando la tecnología anteriormente elegida, JUCE. Ésta es una propuesta del diseño que podría emplear la aplicación, pero no debe considerarse el diseño o propuesta final para el desarrollo del proyecto. El diagrama nos ayuda a concebir una estrategia de trabajo en el momento de preparar la distribución y el plan de trabajo., ya que como más adelante se comenta, la estructura de tareas se centra en la distribución de trabajos sobre la base, componentes y características de la aplicación.



4.2 Paquetes de Trabajo

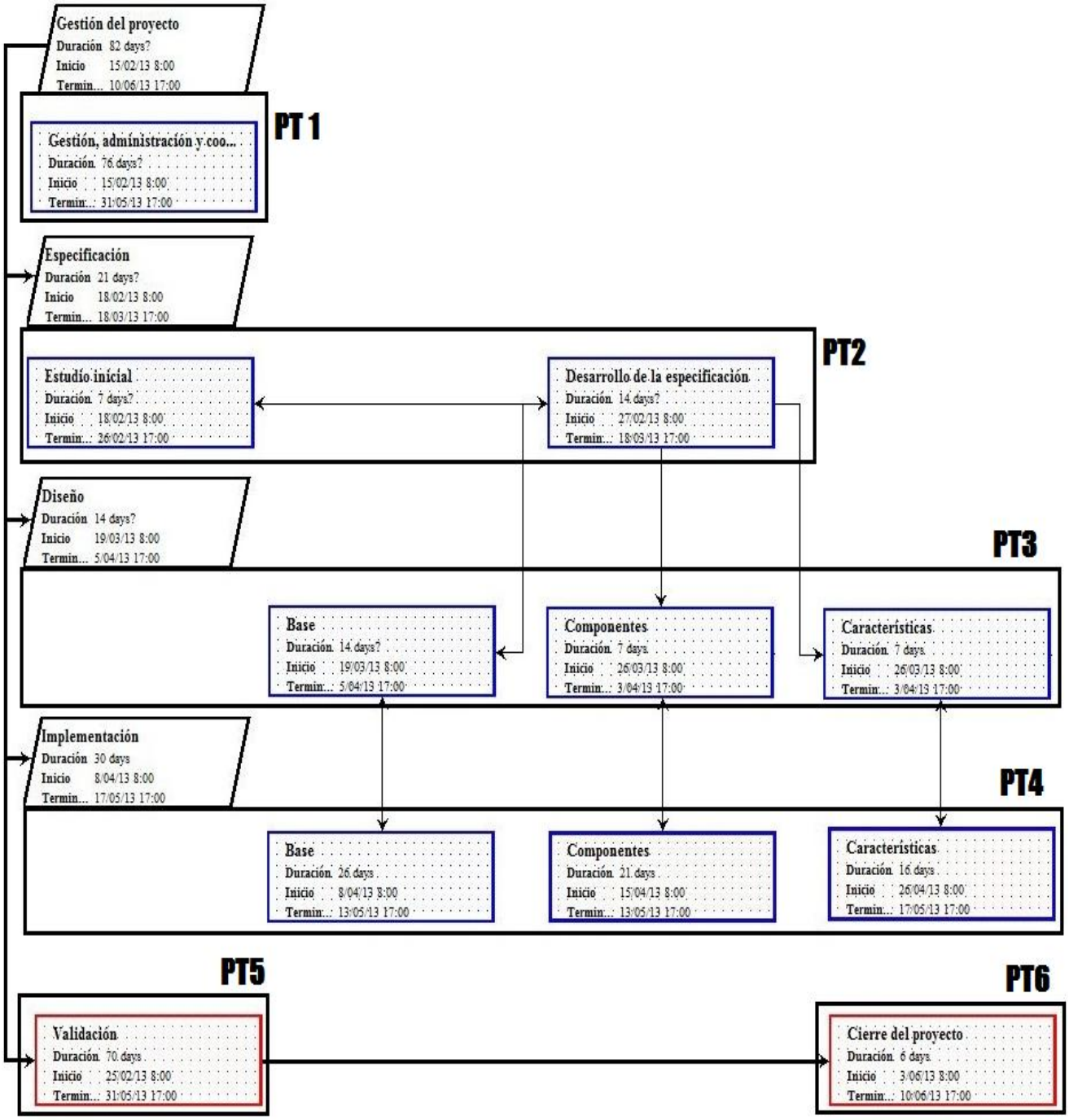
Llegados a esta sección, encontramos la distribución de tareas que engloban la primera aproximación de la propuesta anteriormente vista. Se distribuyen las tareas en paquetes a través del periodo de 12 semanas que dispone el equipo, además de distinguir cada paquete con un factor Implicación o Effort, que indica el trabajo o esfuerzo que debe emplear el equipo en ese paquete. El total de Implicación suma 36, que corresponde a las 12 semanas por el personal fijo del equipo.

La distribución total consta de 6 paquetes de trabajo, a cada cual se le asignan un grupo de tareas. En la columna Paquete tarea se puede distinguir el nombre del paquete y las tareas que engloba, o directamente la tarea principal del paquete.

Nº PT	Paquete Tarea	Semana inicio	Semana final	Implicación
1	Gestión del proyecto: - Gestión, administración y coordinación	S1	S12	4
2	Especificación: - Estudio inicial - Desarrollo de especificación	S1	S3	2
3	Diseño: - Base - Componentes - Características	S3	S6	3
4	Implementación: - Base - Componentes - Características	S4	S11	23
5	- Validación	S3	S12	2
6	- Cierre del proyecto	S11	S12	2

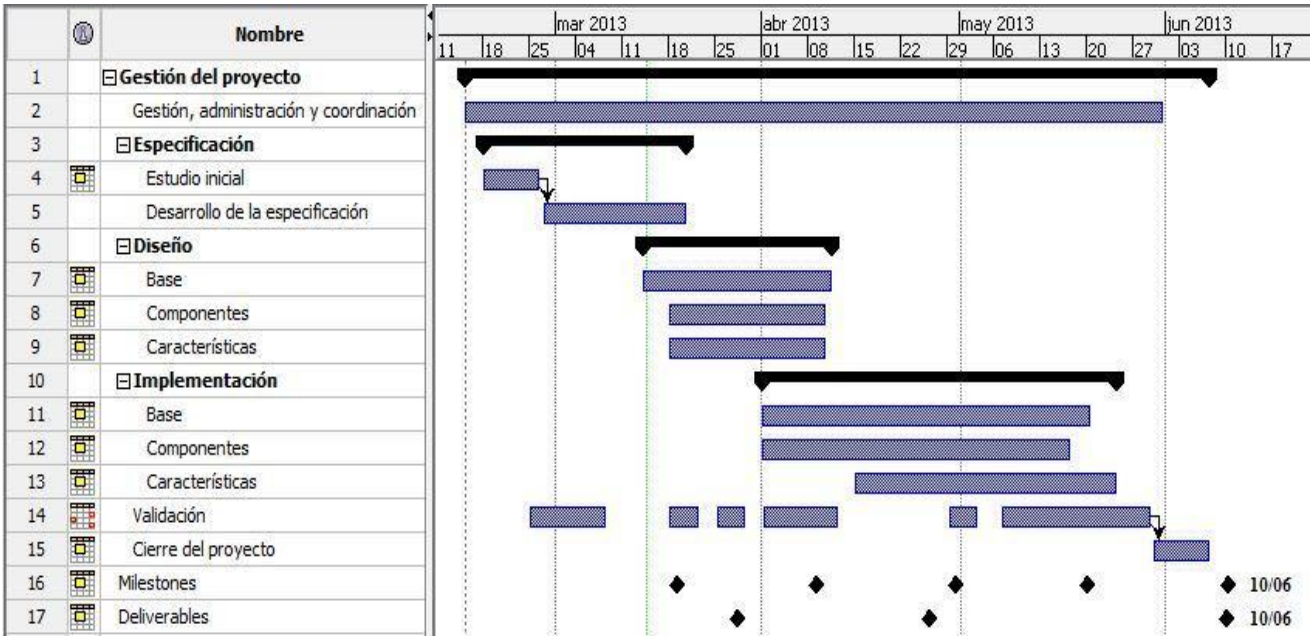
4.3 Diagrama de PERT

La siguiente figura corresponde con el diagrama de PERT del proyecto. En relación con el apartado anterior, Paquetes de tareas, aquí se puede apreciar las conexiones entre los paquetes (y sus tareas) tanto entre paquetes como las tareas que incorporan.



4.4 Diagrama de GANTT

La siguiente figura corresponde con el diagrama de GANTT del proyecto. En relación con el apartado anterior, Paquetes de tareas, aquí se puede apreciar la distribución temporal de cada paquete de trabajo a lo largo de las 12 semanas de duración del proyecto. Para cada paquete de tarea se desglosa las tareas que incluye y se muestran las dependencias y duración dentro del conjunto, así como las fechas de milestones y deliverables.



	Nombre	Duración	Inicio	Terminado	Predecesores
1	**Gestión del proyecto**	81 days?	**15/02/13 8:00**	**7/06/13 17:00**	
2	Gestión, administración y coordinación	76 days?	15/02/13 8:00	31/05/13 17:00	
3	**Especificación**	23 days?	**18/02/13 8:00**	**20/03/13 17:00**	
4	Estudio inicial	7 days?	18/02/13 8:00	26/02/13 17:00	
5	Desarrollo de la especificación	16 days?	27/02/13 8:00	20/03/13 17:00	4
6	**Diseño**	21 days	**14/03/13 8:00**	**11/04/13 17:00**	
7	Base	21 days	14/03/13 8:00	11/04/13 17:00	
8	Componentes	18 days	18/03/13 8:00	10/04/13 17:00	
9	Características	18 days	18/03/13 8:00	10/04/13 17:00	
10	**Implementación**	40 days	**1/04/13 7:00**	**24/05/13 17:00**	
11	Base	36 days	1/04/13 7:00	20/05/13 17:00	
12	Componentes	35 days	1/04/13 8:00	17/05/13 17:00	
13	Características	30 days	15/04/13 8:00	24/05/13 17:00	
14	Validación	68 days	25/02/13 8:00	29/05/13 17:00	
15	Cierre del proyecto	7 days	30/05/13 8:00	7/06/13 17:00	14
16	Milestones	60 days	19/03/13 8:00	10/06/13 17:00	
17	Deliverables	52,75 days	28/03/13 10:00	10/06/13 17:00	

Imágen de zoom a la lista de paquetes de tareas y sus tareas principales, con su duración y predecesores.

4.5 Milestones

En la siguiente tabla aparecen los Milestones que conformarán el proyecto, puntos claves para el seguimiento y control de los avances:

Milestone	Paquete Tarea	Fecha
Milestone 1: Completado de la especificación	PT1 PT2	18 de marzo
Milestone 2: Finalización de la fase de diseño	PT1 PT3, PT4	8 de abril
Milestone 3: Ecuador de la implementación	PT1 PT4, PT5	29 de abril
Milestone 4: Testeo y retoques del prototipo final	PT1 PT4, PT5	20 de mayo
Milestone 5: Finalización del proyecto	PT1, PT6	10 de junio

4.6 Deliverables

En la siguiente tabla aparecen los Deliverables que serán entregados durante la duración del proyecto a nuestro cliente:

Deliverable	Paquete Tarea	Fecha
- Documentación completa de especificación - Primeros diseños de la aplicación - Muestras de tecnología	PT1 PT2 PT3	25 de marzo
- Documentación de diseño - Versión Pre-Alpha de la aplicación con funciones limitadas	PT3 PT4 PT5	25 de abril
- Documentación de implementación - Memoria del proyecto - Versión Beta de la aplicación	PT1 PT4 PT5 PT6	10 de junio

4.7 Personas/Semana

En la siguiente tabla se relacionan el número de personas implicadas en cada paquete de tareas por semana entre las 12 semanas. Como en la tabla de Paquetes de Trabajo, se destaca la semana inicio y final para cada número de paquete, asignando un número aproximado de personas involucradas en ese paquete por semana. No se consideran números decimales, por lo que las personas por semana pueden oscilar a la baja, en el caso de que haya una persona encargada de una tarea una semana mientras otra semana no lo esté, pero en la tabla no se refleja este hecho. Hay que considerar que ciertas tareas se pueden solapar con otras, y que el personal puede atender a distintas tareas en cada semana. Además, el número de personas en una tarea se equilibra con la implicación o esfuerza necesaria por paquete de tarea.

Nº PT	Semana inicio	Semana final	Implicación	Personas/Sem.
1	S1	S12	4	2
2	S1	S3	2	2
3	S3	S6	3	2
4	S4	S11	23	3
5	S3	S12	2	2
6	S11	S12	2	1

5. Propuesta de costes

En este apartado se muestran los conceptos en costes para la viabilidad del proyecto durante el periodo de tiempo especificado anteriormente. Este apartado se divide en dos secciones: Los costes directos y los costes indirectos.

Costes directos

Nóminas

Project Manager (170 h)	(22 €/h) 3.740 €
Analista/Programador (480 h)	(18 €/h) 8.640 €
Programador (400 h)	(15 €/h) 6.000 €

Viajes

Transporte (x7)	(100 €/u) 700 €
Dietas (x7)	(60 €/u) 420 €

Provisión de equipamiento

Hardware

Equipamiento Hardware para desarrollo	3000 €

Software

Visual Studio Professional 2010	615 €
Visual Studio Professional 2010 con MSDN	1.477 €
Licencias Juce (x2)	(891 €/u) 1.782 €
Linux Redhat Enterprise Edition	138 €

Costes indirectos

Mantenimiento instalaciones

Servicio de limpieza (x3)	(300 €/mes)	900 €
Servicio de asistencia técnica		130 €

Comunicaciones

Movistar Fibra Óptica 100Mb (x3)	(60 €/u)	180 €
Comunicación FUSION tlf. y móvil de empresa (x3)	(50 €/u)	150 €

Otros gastos

Luz (x3)	(60 €/u)	180 €
Agua (x3)	(30 €/u)	90 €
Mantenimiento y otros conceptos (x3)	(40 €/u)	120 €

Seguros (Trimestral)

Generali Seguros (Garantías básicas + Robo)	204 €
Seguro de fiabilidad (Enterprise Development Liability) *- Pérdida de Documentos* *- Intromisión ilegítima* *- Protección de datos*	349 €
Seguro de cobertura laboral	280 €
Seguro de responsabilidad profesional: *- Gastos de Defensa* *- Fianzas Judiciales*	250 €

6. Análisis de riesgos y acciones correctivas

En este apartado se muestran los principales riesgos asegurables que están presentes desde el inicio del proyecto (incluyendo las negociaciones iniciales y la firma de contratos) hasta la finalización del proyecto y futuros sucesos y/o implicaciones del producto. A continuación se muestran los principales riesgos a tener en cuenta de una forma generalizada, mientras que a posterior se declaran los riesgos concretos con las acciones a tomar:

Cambios en las condiciones del mercado: Para el transcurso de tiempo estipulado para el desarrollo del proyecto cabe tener en cuenta otros productos que pudiesen aparecer en el mercado como competidores directos, para ello se tomarían acciones rectificatorias incluyendo estudios del mercado y la posible renegociación del contrato entre cliente / desarrolladora.

Accidentes y daños materiales: Para los posibles daños que afecten a las instalaciones o el material necesario para llevar a cabo el desarrollo del proyecto se cuenta con un seguro de cobertura.

Accidentes y daños personales: Para los posibles daños que afecten el personal involucrado en el proyecto, ya sea por accidente laboral o extralaboral se cuenta con un seguro de cobertura que permita solventar este tipo de sucesos.

Modificación de política y efectos legales: Existen métodos que se encargan de cubrir la necesidad de mantener dentro de la legalidad todo desarrollo tanto tecnológico (posibles patentes y/o licencias) así como el cumplimiento de los términos acordados para el producto final, tomando acciones sujetas a leyes y su actuación pertinente.

Cumplimientos de términos y contratos: Cobertura jurídica que protege y cubre el cumplimiento y gastos que puedan comportar tales sucesos, quedando sujeto a ley con las acciones pertinentes.

Errores, robo o intrusión externa: En el caso de posibles errores que puedan afectar al desarrollo del producto final existe una cobertura que permite proteger el desarrollo del proyecto sin implicar pérdidas para el cliente. En el caso de espionaje externo o robo de material se mantiene la cobertura total y metodologías jurídicas para perseguir tales sucesos tomando medidas sujetas a ley.

En la siguiente tabla se muestra el análisis de los riesgos según los sucesos a tener cuenta, clasificados según el tipo de impacto y las medidas oportunas a tomar en caso de la materialización de tales sucesos:

Riesgo	Tipo	Acción
Surge nuevo producto competidor en el mercado	Medio	Estudio de nuevas vías de desarrollo y modificación del producto
Nuevas tecnologías que dejan obsoletas las actuales	Alto	Estudio de las nuevas tecnologías y adaptación del producto
Descontinuación del software base	Medio	Estudio de posibles alternativas o desarrollo propietario del software base
Daños materiales de infraestructura o instalaciones	Alto	Seguro de cobertura ante accidentes o desastres, posible reemplazo del suceso en cuestión
Daños materiales del equipamiento	Medio	Seguro de cobertura para daños de equipamiento y reemplazo necesario
Accidente laboral	Medio	Seguro de riesgos laborales, y posibilidad de contratación de sustitución
Bajas laborales	Medio	Seguro de cobertura de personal, en caso de gravedad proceder a nuevas contrataciones
Modificación de políticas	Medio	Reconsideración de alternativas y adaptación de nuevas políticas
Modificación de acuerdos de licencia	Medio	Renegociación sobre los acuerdos, adquisición de nuevas licencias o toma de medidas legales

Riesgo	Tipo	Acción
Incumplimiento de patentes	Alto	Estudio de patentes violadas, y búsqueda de posibles alternativas. En última instancia proponer acuerdos de beneficio mutuo
Incumplimiento de términos	Alto	Diálogo y negociación entre las partes implicadas, proponiendo nuevos términos. En caso negativo tomar medidas y soluciones legales
Incumplimento de contratos	Alto	Renegociación amistosa sobre acuerdos y contratos, o tomar medidas legales ante incumplimientos y/o omisión de los acuerdos.
Errores o negligencias	Medio	Seguro de cobertura de fiabilidad (liability) que responde ante errores, que puede cubrir costes, con el consiguiente estudio de soluciones y rectificaciones
Intrusión o robo	Alto	Refuerzo de las medidas de protección del material de propiedad intelectual, y consiguiente investigación ante el suceso en cuestión a través de medidas legales

Muchos de estos riesgos se han considerado de carácter general y básicos para la cobertura del proyecto, sin embargo no son la totalidad en previsión del proyecto, limitados ni excluyentes de otros no citados en esta documentación.

7. Indicadores y seguimiento

En este apartado se muestran las tablas con los indicadores (KPI) para el seguimiento y control de todos los objetivos principales del proyecto de manera muy granular, claves para el cumplimiento de dichos objetivos.

Objetivo 1: Edición de GUI	**Key Performance Indicators**
Entorno de trabajo destinado a la edición de interfaz gráfica para el usuario. (*) Se entiende componente elementos tales pero no limitado a como ventanas, textos, botones, barras, etc.	**KPI-1:** Entorno mínimo de trabajo funcional que permita dibujar componentes* propios de una interfaz gráfica de usuario, a modo de arrastrar y soltar. **Milestone 3** **KPI-2:** Capacidad funcional de distribución (mover), ajuste (tamaño) y edición (atributos) de componentes* por el espacio destinado a la construcción de interfaces. **Milestone 3** **KPI-3:** Función de deshacer y rehacer los cambios realizados sobre los componentes. **Milestone 4**

Objetivo 2: Versátil	**Key Performance Indicators**
Editor de interfaces con capacidad de diseñar distintos tipos de interfaces gráficas orientadas a propósitos de temática* muy concreta. (*) Se considera temática concreta cuando se habla de una interfaz gráfica para un programa destinado a, por ejemplo, ofimática, TIC, dibujo y renderizado, etc.	**KPI-4:** Listado de opciones de selección de plantillas al iniciar un nuevo proyecto. **Milestone 5** **KPI-5:** Integración de plantillas orientadas a propósitos de temática muy concreta que presenten componentes y atributos propios de la temática seleccionada. **Milestone 5** **KPI-6:** Capacidad de importar/exportar nuevas plantillas en forma de archivo, a modo de incluir nuevas plantillas de inicio de proyecto. **Milestone 5**

Objetivo 3: Modular y extensible	Key Performance Indicators
Posibilidad de incluir nuevos componentes y funcionalidades, en forma de módulos, directamente sobre la aplicación, y que permita al desarrollador la capacidad de extender el conjunto en un futuro si es deseado.	**KPI-7:** Función que permite cargar en forma de módulos, nuevas funciones directamente desde archivos. **Milestone 5** **KPI-8:** Al tratarse de código abierto, permitir la extensión del programa con la posibilidad de incluir nuevas funcionalidades sin modificar el núcleo de la aplicación. **Milestone 5**

Objetivo 4: Independiente	Key Performance Indicators
La herramienta debe de ser capaz de funcionar como aplicación stand-alone.	**KPI-9:** La aplicación debe correr a través del ejecutable, sin ningún tipo de dependencia de librerías externas, códecs o drivers. **Milestone 3** **KPI-10:** La aplicación es capaz de ejecutarse en sistemas operativos Windows (la compatibilidad con otros SO aún está por decidir). **Milestone 3**

Objetivo 5: Personalizable	Key Performance Indicators
Capacidad de adaptar la apariencia del entorno y sus componentes	**KPI-11:** Opción de selección de apariencia de un componente a través de sus propiedades. **Milestone 3** **KPI-12:** Opción de alteración de apariencia, color, y formas mediante atributos de componente. **Milestone 3**

Objetivo 6: Importación y exportación	Key Performance Indicators
La herramienta debe permitir extraer el desarrollo en curso del proyecto de su interfaz a ciertos formatos, así mismo como importar y adaptar proyectos previamente desarrollados.	**KPI-13:** Funcionalidad de exportación de proyecto en XML a un archivo (otros tipos de conversión posibles). **Milestone 4** **KPI-14:** Funcionalidad de importar un proyecto a partir de un archivo en XML (otros tipos de conversión posibles). **Milestone 4** **KPI-15:** Guardar el estado actual de un proyecto, y exportarlo para la edición de código. **Milestone 4**

Objetivo 7: Caja de herramientas	Key Performance Indicators
Crear un entorno o marco de trabajo que integre conjuntos de herramientas, a base de módulos y opciones para la creación de todo tipo de componentes	**KPI-16:** Espacio de trabajo dividido en zonas por función. **Milestone 3** **KPI-17:** Zona de selección de herramientas mostradas visualmente a través de iconos. **Milestone 3** **KPI-18:** Zona de visualización y edición de propiedades y atributos de los componentes seleccionados. **Milestone 3** **KPI-19:** Barra de menú principal del menú, que integra opciones básicas de guardado, exportación e importación, entre otras múltiples opciones. **Milestone 4**

Objetivo 8: Las herramientas	Key Performance Indicators
Herramientas* con la capacidad de introducir componentes en el espacio de trabajo, principalmente para cubrir cualquier necesidad de desarrollo. Las herramientas forman parte de la caja de herramientas. (*)Una herramienta se define como la función que crea el componente correspondiente en la zona de trabajo.	**KPI-20:** Herramientas básicas de interfaz gráfica que incluyen componentes como ventana, etiqueta de texto e imagen. **Milestone 3** **KPI-21:** Componentes básicos como botones y sliders . **Milestone 3** **KPI-22:** Componentes contenedores básicos como combobox, y listbox. **Milestone 4** **KPI-23:** Componentes extras. **Milestone 4** **KPI-24:** Activación de encasillado y recalibración de escala de encasillado. **Milestone 3**

Objetivo 9: Edición de propiedades	Key Performance Indicators
Acceso a los parámetros y atributos de cada módulo o componente para visualizar, editar o rectificar en tiempo real.	**KPI-25:** Campo de visualización de atributos funcional y editable. **Milestone 3** **KPI-26:** Funciones de componente activables/desactivables. **Milestone 3** **KPI-27:** Listado de parámetros según atributos desplegable y funcional (componentes que lo soporten). **Milestone 3**

8. Referencias

En este apartado aparecen los enlaces de referencia a información sobre las tecnologías de JUCE propuestas y otros enlaces de información de interés.

JUCE user forum - Raw Material Software: http://rawmaterialsoftware.com/juce.php

JUCE Cross-Platform C++ Library: http://www.juce.com/

julianstorer/JUCE - GitHub: https://github.com/julianstorer/JUCE

JUCE Interface Designer Page: http://theadd.github.io/juced/