



## **2 Elaboración de interfaces mediante documentos XML**

## Sumario

2	Elaboración de interfaces mediante documentos XML.....	1
2.1	Convenciones empleadas.....	5
2.2	Introducción.....	6
2.3	Lenguajes basados en XML.....	6
2.3.1	Etiquetas.....	7
2.3.2	Atributos y valores.....	8
2.4	Lenguajes de descripción de interfaces basados en XML.....	9
2.4.1	Proceso de elaboración de las interfaces.....	10
2.4.2	XAML.....	11
2.4.3	UIML.....	11
2.4.4	XIML.....	13
2.4.5	SVG (Scalable Vector Graphics).....	14
2.5	Herramientas para la creación de interfaces multiplataforma.....	14
2.5.1	Presentación de algunas herramientas.....	15
2.6	Blend para Visual Studio.....	16
2.6.1	Controles. Contenedores.....	20
2.6.2	Controles y propiedades.....	21
2.7	Glade.....	23
2.7.1	Paletas y vistas.....	24
2.7.2	Contenedores de controles.....	24
2.7.3	Controles y propiedades. Alineación, tamaño y ubicación de los controles.....	25
2.8	Scene Builder.....	27
2.9	Eventos. Secuencia de eventos.....	28
2.10	Análisis y edición de documentos XML.....	28
2.11	Ejemplo de desarrollo de una interfaz utilizando XML.....	29
2.11.1	Descripción del problema.....	31
2.11.2	Creación proyecto JavaFX.....	32
2.11.3	Edición de la interfaz.....	34
2.11.4	Implementación del código fuente.....	37










## Índice de figuras

Figura 1: Esquema en forma de árbol jerárquico de un documento XML genérico.....	7
Figura 2: Esquema en forma de árbol jerárquico de un documento XML genérico.....	8
Figura 3: Diagrama de mapeado de una interfaz con XML.....	10
Figura 4: Código básico para crear una ventana en XAML.....	11
Figura 5: Código básico en lenguaje UIML para crear una ventana de diálogo simple.....	12
Figura 6: Funcionamiento de la tecnología XIML.....	13
Figura 7: Esquema de uso de una herramienta para el desarrollo de interfaces gráficas..	16
Figura 8: Instalación de Visual Studio Community.....	17
Figura 9: Pantalla de acceso a Blend Studio.....	18
Figura 10: Entorno de trabajo de Blend Studio.....	20
Figura 11: Entorno de trabajo de Blend Studio.....	21
Figura 12: Contenedores de Blend Studio.....	21
Figura 13: Botón en Blend Studio.....	22
Figura 14: Etiqueta en Blend Studio.....	22
Figura 15: Propiedades de los controles en Blend Studio.....	23
Figura 16: Controles de la aplicación Glade.....	24
Figura 17: Contenedores de la aplicación Glade.....	25
Figura 18: Panel de propiedades de Glade.....	26
Figura 19: Interfaz de la aplicación Scene Builder.....	27
Figura 20: Gluon Scene Builder.....	30
Figura 21: Configuración en NetBeans de JavaFX.....	31
Figura 22: Creación del proyecto Java FX.....	32
Figura 23: Plantilla del proyecto Java FX.....	33
Figura 24: Vista de la aplicación de ejemplo.....	34
Figura 25: Edición del fichero FXMLDocument dentro de la aplicación Scene Builder.....	35
Figura 26: Creación de la vista del proyecto.....	36
Figura 27: Configuración de elementos.....	37

Material docente elaborado a partir de la base de los materiales formativos de FP en línea propiedad del Ministerio de Educación e Formación Profesional.

[Aviso Legal](#)

## 2.1 Convenciones empleadas

	Notas de introducción
	Aclaración
	Archivos de configuración, de registro...
	Casos de uso
	Código fuente
	Avisos o advertencias
	Capturas de pantalla, imágenes
	Actividades
	Enlace recomendado

	Iconos proporcionados por Papyrus Development Team <a href="https://github.com/PapyrusDevelopmentTeam/papyrus-icon-theme">https://github.com/PapyrusDevelopmentTeam/papyrus-icon-theme</a>
---	---

## 2.2 Introducción



XML (eXtensible Markup Language o Lenguaje de Marcado eXtensible) es un lenguaje basado en texto de definición de documentos, que permite representar información estructurada de gran variedad de documentos.

## 2.3 Lenguajes basados en XML.

Al estar puramente basado en texto, un documento XML será un archivo plano en el que, a través de una serie de elementos propios se definen aspectos del documento a generar, pero no de su diseño propiamente dicho, sino de su estructura.

Hacer independiente la estructura de un documento de la constante evolución de las herramientas que se usan para visualizarlo ha sido, desde hace tiempo, foco de interés de los desarrolladores y desarrolladoras de aplicaciones informáticas. En los años sesenta, Charles F. Goldfarb, por encargo de IBM desarrolló el lenguaje GML, Generalized Markup Language, cuyo objetivo era describir la estructura de los documentos de forma que el resultado no dependiese de una determinada plataforma o una aplicación específica. De la evolución de GML surgió SGML (Standard Generalized Markup Language) que se convirtió en el estándar internacional ISO 8879. Sin embargo, tuvo difusión tan sólo en medios académicos y de la administración, pero no tuvo demasiado éxito entre los usuarios medios debido a su dificultad.

El hito que supuso la amplia difusión de las tecnologías de la comunicación y la circulación de documentos electrónicos entre cualquier punto del mundo fue la creación de la World Wide Web (Tim Berners-Lee, en 1990), y, asociado a ello la aparición de HTML, lenguaje de descripción de documentos basado en SGML para la web.

HTML es un lenguaje que permite combinar en un solo documento el contenido con el formato, por ejemplo:

Si en un documento HTML se añade el siguiente código:



```
El padre del lenguaje <b>GML</b> fue <b>Charles F. Goldfarb</b>
```

Se conseguirá algo como esto:



```
El padre del lenguaje GML fue Charles F. Goldfarb
```

Ya que encerrar un texto entre `<b>` y `</b>` equivale a aplicar el formato negrita. Estos elementos de HTML como `<b>` y `</b>` se denominan etiquetas, cada etiqueta tiene un significado en el diseño del documento final o en su estructura.

XML sin embargo, no incluye ninguna orientación al diseño es puramente un lenguaje estructural cuyo objetivo es definir cómo se organiza la información en un documento, el

diseño se definirá después, de esta forma se hace independiente al documento de la plataforma, e incluso de la aplicación con la que se va a visualizar (Web, impreso, como documento de texto, y más). Por ejemplo, si definimos una página web con XML podremos verla fácilmente en un navegador web, en un teléfono móvil, o, incluso, en un lector Braille, sin modificar el documento de base. Lo que cambia es la forma en que se interpreta el contenido.

XML nace con el objetivo de convertirse en un estándar para el intercambio de información estructurada entre diferentes plataformas: Windows, Android, Linux, iOS, etc. Se puede utilizar en bases de datos, editores de texto, cálculos, dispositivos móviles, etc... Actualmente es la base para la creación de interfaces gráficas de muchos generadores de códigos o IDEs como Visual Studio, Android Studio, NetBeans y Eclipse.

### 2.3.1 Etiquetas

XML es un tanto parecido a HTML en el sentido de que utiliza etiquetas para definir elementos dentro de una estructura, con la diferencia de que cuando usamos HTML se trabaja con una serie de etiquetas predefinidas, y en XML se pueden definir según convenga a las necesidades del proyecto.



Esquema en forma de árbol jerárquico de un documento XML genérico

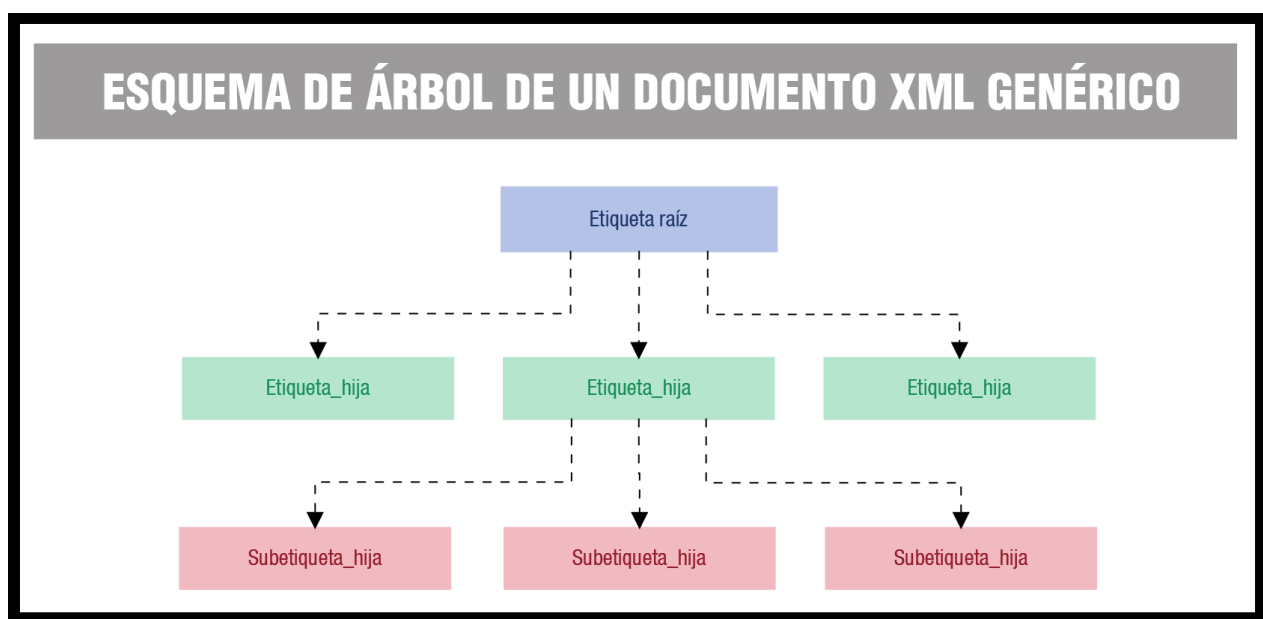


Figura 1: Esquema en forma de árbol jerárquico de un documento XML genérico

Ministerio de Educación y Formación Profesional. (Elaboración propia)

Las etiquetas en un documento XML tienen siempre **etiqueta de cierre**, de manera que toda la información referente al elemento queda comprendida entre ambas.

Siempre existe una **etiqueta raíz**, que hace referencia al tipo de objeto que se describe. El resto son características o elementos del objeto, y se colocan anidadas dentro de la

etiqueta raíz como etiquetas hijas. Por eso decimos que un documento XML tiene **estructura de árbol**.

Las etiquetas en un documento XML tienen siempre etiqueta de cierre, de manera que toda la información referente al elemento queda comprendida entre ambas.

Siempre existe una etiqueta raíz, que hace referencia al tipo de objeto que se describe. El resto son características o elementos del objeto, y se colocan anidadas dentro de la etiqueta raíz como etiquetas hijas. Por eso decimos que un documento XML tiene estructura de árbol.



### Estructura de árbol de un documento XML genérico

```
<Etiqueta_raiz>
  <etiqueta_hija>
    <subetiqueta_hija>...</subetiqueta_hija>
    ...
  </etiqueta_hija>
</Etiqueta_raiz>
```



### Esquema en forma de árbol jerárquico de un documento XML genérico

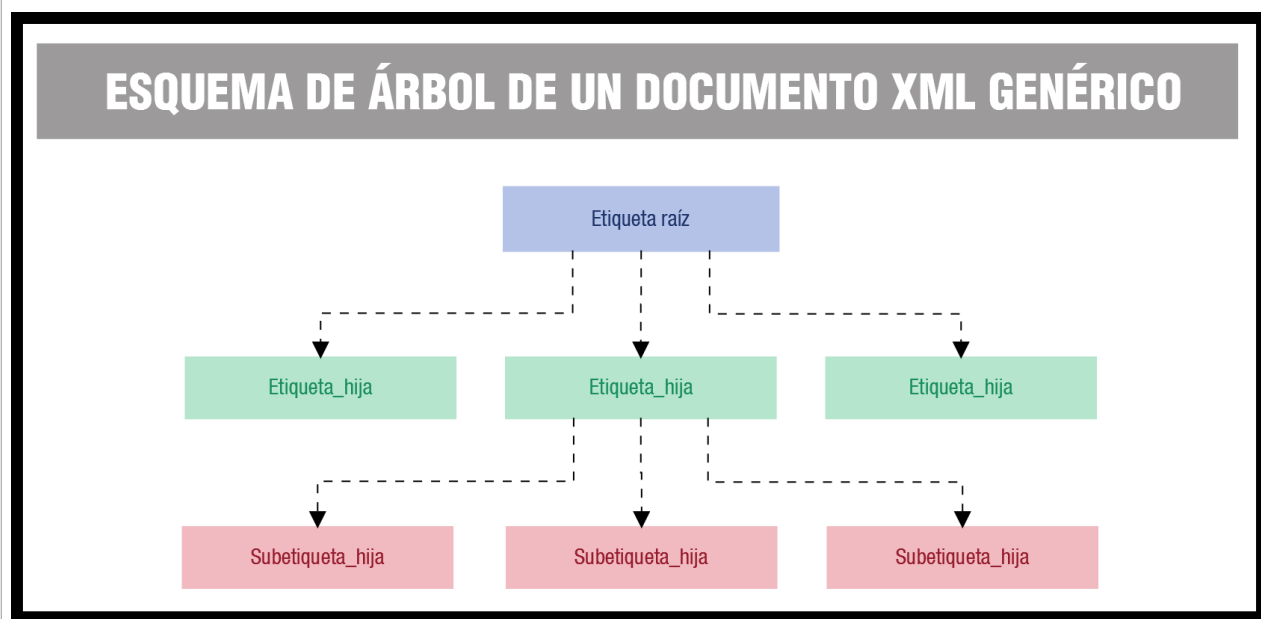


Figura 2: Esquema en forma de árbol jerárquico de un documento XML genérico

Ministerio de Educación y Formación Profesional. (Elaboración propia)

## 2.3.2 Atributos y valores

Se dice que un elemento XML tiene contenido cuando se ha añadido algún texto entre la etiqueta de apertura y cierre. Sin embargo, pueden darse casos en los que no se pueda almacenar toda la información pertinente del contenido sólo en el texto que encierran las etiquetas.



Cuando necesitamos añadir información adicional a un elemento XML de alguna manera modificamos la etiqueta añadiéndole **atributos**.



```
<etiqueta atributo="valor">Contenido</etiqueta>
```

Los atributos permiten proporcionar información adicional sobre el elemento. Por ejemplo, puedo precisar si el teléfono que he guardado para mi amigo Javier es su teléfono fijo, su móvil o el teléfono del trabajo añadiendo el atributo tipo:

No siempre ocurre que cumpliendo los requisitos anteriormente comentados el documento XML sea correcto. Para asegurarnos de que un documento XML esté bien formado debe cumplir las siguientes reglas:

- Debe existir un elemento raíz.
- Todos los elementos XML deben tener su correspondiente etiqueta de cierre.
- Es sensible a mayúsculas.
- El anidamiento debe hacerse conforme a la estructura del árbol del documento. Es decir, si abro la etiqueta y a continuación se cierran en sentido inverso, no puedo cerrar antes y después
- Los valores de los atributos van entrecomillados siempre.

A la hora de poner nombre a las etiquetas se deben seguir las siguientes recomendaciones:

- Se pueden usar letras, número y otros caracteres.
- No se puede empezar por un número o signo de puntuación.
- No se puede empezar por las letras XML .
- Los nombres no pueden contener espacios en blanco.

## 2.4 Lenguajes de descripción de interfaces basados en XML

El desarrollo de interfaces de usuario es una actividad compleja que requiere de la intervención de múltiples factores. El desarrollador o desarrolladora se enfrenta a una serie de hitos que debe superar:

- En primer lugar, se debe definir lo que se espera de la interfaz, que requisitos debe cubrir y el entorno en el que se integra.
- También se debe tener en cuenta la variedad de herramientas y lenguajes de programación que exigen un elevado nivel de conocimientos.
- Por otra, parte necesitamos interfaces (así como las aplicaciones subyacentes) para múltiples contextos de uso, que dependen del usuario (por su idioma, preferencias o posibles discapacidades), de la plataforma (ordenadores

personales, móviles, miniportátiles, pantallas, etc.), del modo de presentación de la interfaz (gráfica o texto) o del dispositivo final (pantallas de ordenador, pizarras digitales, proyectores, etc.).

### 2.4.1 Proceso de elaboración de las interfaces

La primera posibilidad para desarrollar una interfaz suele ser emplear el mismo lenguaje, normalmente de alto nivel, que se usa para implementar la funcionalidad de la aplicación como Java, C#, etc. Tiene como ventaja la sencillez, puesto que no es preciso adquirir nuevos conocimientos, la creación de la interfaz se integra en el proceso de desarrollo de la aplicación y, normalmente, no es necesario trabajar con herramientas diferentes, ni en la programación ni posteriormente a la hora de compilar o ejecutar el programa. Sin embargo, tiene como principal desventaja la dependencia de la plataforma, del dispositivo y del propio lenguaje. Básicamente, la portabilidad de las interfaces creadas de esta manera serán las que proporcione el lenguaje.

La creación de interfaces de usuario usando lenguajes de descripción basados en XML pretende solventar esto, proporcionando un medio que permita construir interfaces mediante descripciones de alto nivel en los distintos aspectos de la interfaz: estructura y comportamiento, de modo que a partir de la descripción se pueda generar de forma automatizada la interfaz de usuario final. Además, este proceso permite centrar el desarrollo en las necesidades del usuario, puesto que no se realiza siguiendo las directrices marcadas por el lenguaje.



Diagrama de mapeado de una interfaz con XML

#### DIAGRAMA DE MAPEADO DE UNA INTERFAZ CON XML

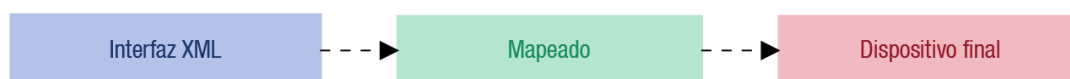


Figura 3: Diagrama de mapeado de una interfaz con XML

En este ámbito, se utilizará una notación de alto nivel para desarrollar la interfaz, que se almacena en un archivo aparte, en formato XML. Posteriormente se trata este archivo para obtener el código de la interfaz que se integrará en la aplicación. A este procedimiento se le denomina mapear la interfaz. El proceso de mapeado depende del lenguaje concreto que se esté usando.

Tras el correspondiente proceso se podrá visualizar la interfaz en un dispositivo final.

A continuación, veremos algunos ejemplos de lenguajes para el desarrollo de interfaces basados en XML y como realizan el proceso de mapeado.

### 2.4.2 XAML

Es un lenguaje de marcas empleado para la creación de interfaces en el modelo de programación .NET Framework de Microsoft. Las aplicaciones creadas podrán ejecutarse en entornos Windows.

Mediante XAML se puede crear lo que se conoce como RIA (Rich Interface Applications) que contienen abundante material gráfico. XAML consta de una serie de elementos XML para representar los principales componentes gráficos, así como la distribución, paneles y manejadores de eventos. Puede hacerse programando directamente la interfaz mediante un editor de texto, o mediante el entorno de desarrollo gráfico incluido en la herramienta Expression Blend. El código asociado a la interfaz es puramente declarativo, es decir, hace referencia tan solo al aspecto visual, pero no añade funcionalidad, salvo ciertas respuestas muy sencillas a interacciones con el usuario.

La realización de cálculos se añade en un archivo independiente. Esto permite al equipo de desarrollo y al de diseño trabajar por separado, sin interferir mutuamente en su trabajo.



Código básico para crear una ventana en XAML con un botón que diga Haz Click!.

```
<Page
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
  presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  x:Class="EspacioEjemplo.PaginaEjemplo">
  <Button Click="Accion_Click" >Haz Click!</Button>
</Page>
```

*Figura 4: Código básico para crear una ventana en XAML*

María José Navascués González (Elaboración propia)

**Proceso de mapeado:** se realiza en tiempo de ejecución, los elementos de la interfaz se conectan con objetos del framework .NET y los atributos con propiedades de estos objetos para integrarlos en la aplicación. Para facilitar la traducción de XAML a código .NET se ha creado una relación para cada elemento XAML con una clase de la plataforma .NET.

### 2.4.3 UIML

Este lenguaje permite generar interfaces que son independientes de la plataforma y el lenguaje subyacente. Para generar una interfaz UIML se precisa crear un documento XML con la definición de la interfaz, utilizando los elementos de UIML y una hoja de estilos que traslade esa definición a la plataforma y lenguaje seleccionado, de esta forma para una aplicación concreta tan solo necesitaremos un documento UIML, y tantas hojas de estilo para la traducción como nos sean necesarias.

Este sistema tiene como principal ventaja que solo se precisa un único diseño de la interfaz de la aplicación, independientemente del dispositivo donde será visualizado, con lo que evitamos el peligro de desarrollar interfaces para dispositivos que no estén en el mercado en el futuro.

UIML describe una interfaz en tres niveles: presentación, contenido y lógica.

La presentación se refiere a la apariencia de la interfaz; el contenido se refiere a los componentes de la interfaz y la lógica se refiere a la interacción usuario – interfaz (eventos del ratón, teclado...).

En UIML, una interfaz de usuario es una jerarquía de elementos XML. Cada uno de los componentes de una interfaz (botones, cajas de texto, etiquetas...) son una entidad XML. Estas entidades son elementos **Part**. Cada uno tiene asociado un elemento, que puede ser texto, imagen, etc.

Para definir el **comportamiento** de la interfaz se realiza el mapeo de las partes a los elementos correspondientes del lenguaje de implementación elegido y la conexión con la lógica de aplicación.



Código básico en lenguaje UIML para crear una ventana de diálogo simple, con una etiqueta y un botón

```
<UIML>
<HEAD>
  <AUTHOR>María José Navascués</AUTHOR>
  <DATE>1 Mayo, 2011</DATE>
  <VERSION>1.0</VERSION>
</HEAD>
<APP CLASS="App" NAME="Dialogo">
  <GROUP CLASS="Dialog" NAME="DialogoConBoton">
    <ELEM CLASS="DialogMessage" NAME="Mensaje"/>
    <ELEM CLASS="DialogButton" NAME="Boton_OK"/>
  </GROUP>
</APP>
<DEFINE NAME="Boton_OK">
  <PROPERTIES>
    <ACTION
      VALUE="Dialogo.EXISTS=false"
      TRIGGER="Selected"
    />
  </PROPERTIES>
</DEFINE>
</UIML>
```

*Figura 5: Código básico en lenguaje UIML para crear una ventana de diálogo simple*

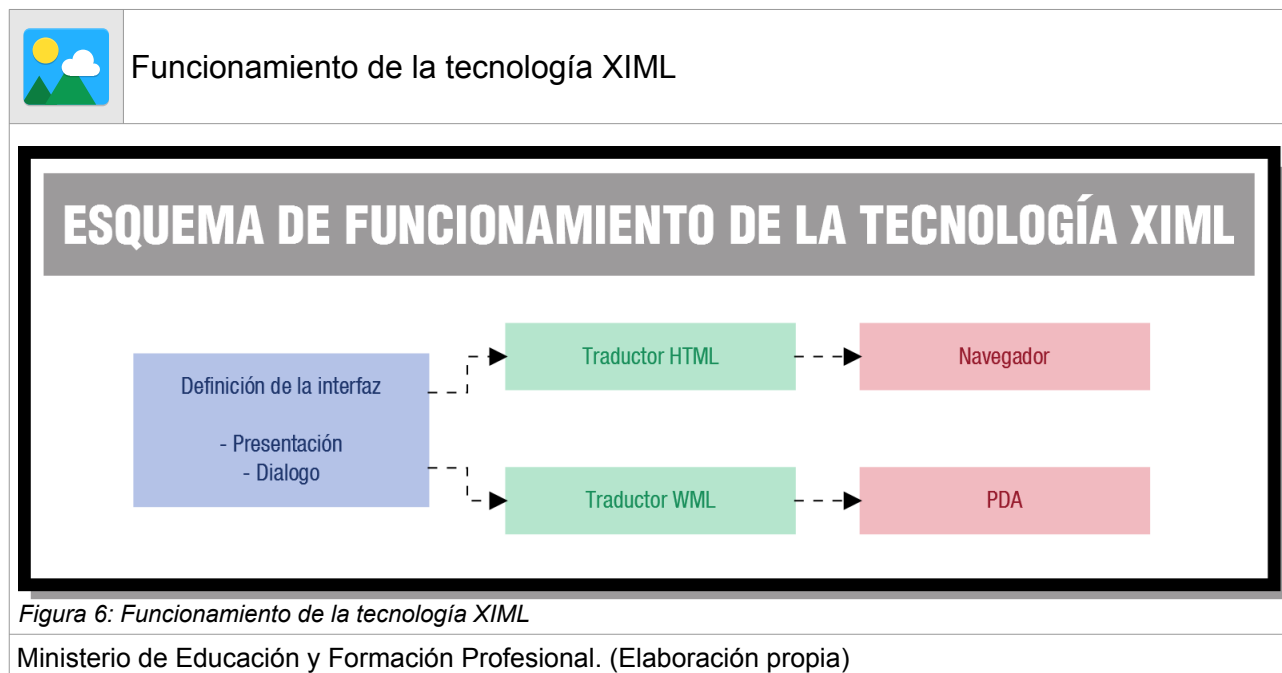
María José Navascués González (Elaboración propia)

Estos bloques facilitan la separación entre los elementos que componen la interfaz, distinguiendo entre el modelado estructural, la visualización y el modelado del comportamiento.

El lenguaje UIML permite la traducción automática al lenguaje utilizado por el dispositivo final. El proceso de traducción se realiza en el propio dispositivo o en el servidor de la interfaz dependiendo del dispositivo del que se trate.

#### 2.4.4 XIML

Es un lenguaje de desarrollo de interfaces cuyo objetivo es cubrir todo el ciclo de vida del software, incluyendo las fases de diseño, operación y evaluación, por lo tanto, además de proporcionar la infraestructura para poder diseñar una interfaz gráfica de usuario con cierto nivel de complejidad, también proporciona herramientas para atender a todo el proceso de desarrollo de la misma.



XIML trabaja con los elementos abstractos y concretos de una interfaz de usuario. Los elementos abstractos hacen referencia al contexto en el que se interacciona con la interfaz, y se representan en:

- **Tareas:** procesos o tareas de usuario que puede ejecutar la interfaz.
- **Dominio:** conjunto de objetos y clases con los que se opera desde la interfaz. Están distribuidos jerárquicamente.
- **Usuarios:** también se organizan jerárquicamente y representan usuarios o grupos de usuarios que hacen uso de los datos y procesos en las tareas.

Los elementos concretos hacen referencia a aquello que se representa físicamente en la interfaz, como los controles de formulario, y se representan en:

- **Presentación:** colección jerárquica de elementos que interaccionan con el usuario desde la interfaz, puede ser desde una ventana a un botón, pasando por controles más complejos como un control ActiveX.

- **Dialogo:** colección jerárquica de acciones de interacción que permiten al usuario comunicarse con los elementos de la interfaz.

Los elementos de la interfaz interaccionan a través de relaciones en las que se indica que elementos intervienen y el tipo de relación que hay entre ambos.

**Proceso de mapeado:** En este lenguaje se separa completamente la definición de la interfaz de la forma en que es renderizada. Precisa de un traductor que traslade la definición a código visible para cada dispositivo específico en el que se quiera usar la interfaz.

De esta forma se consigue un lenguaje completamente **multiplataforma**.

## 2.4.5 SVG (Scalable Vector Graphics)

Los gráficos vectoriales redimensionables o SVG son una especificación para describir gráficos vectoriales bidimensionales. Estos gráficos pueden ser animados o estáticos. Poseen un formato XML. Actualmente la mayoría de los navegadores soporta SVG.

Hay que destacar que las imágenes vectoriales pueden ser redimensionadas sin perder calidad. No ocurre lo mismo con las imágenes en mapa de bits.

SVG permite tres tipos de gráficos:

- Elementos geométricos vectoriales: líneas, círculos, rectas, etc.
- Imágenes en mapa de bits.
- Textos.
- Los objetos gráficos pueden ser agrupados, transformados y compuestos en objetos previamente renderizados. Se les puede aplicar los mismos estilos a cada grupo.

## 2.5 Herramientas para la creación de interfaces multiplataforma

Existen muchos tipos de software para la creación de interfaces de usuario. Habitualmente estas herramientas tienen en común que para generar interfaces gráficas usan un sistema de ventanas, las cuales permiten la división de la pantalla en diferentes regiones rectangulares, llamadas "ventanas" cuya parte central es el conjunto de herramientas (toolkit).

Cuando creamos una aplicación con interfaces gráficas, en primer lugar, hay que **diseñar la interfaz**, normalmente el toolkit contiene los objetos gráficos más empleados, tales como menús, botones, etiquetas, barras de scroll, y campos para entrada de texto que compondrán la interfaz, mientras que el sistema de ventanas provee de procedimientos que permiten dibujar figuras en la pantalla y sirve como medio de entrada de las acciones del usuario. En la imagen puedes apreciar los conjuntos de herramientas enmarcados en rojo (en la imagen cuadro derecho y barra de herramientas superior) y la zona de dibujo

en azul (en la imagen cuadro central). En algunas aplicaciones denominan **Widgets** a los controles o elementos que podemos añadir a una interfaz.

A continuación, se **añade funcionalidad** a los elementos de la interfaz a través de una serie de procedimientos definidos por el programador o programadora. La función de estos procedimientos es el decidir la forma en que se comportarán los objetos gráficos.

Por último, **se conecta la interfaz generada con la aplicación de destino**. Esto se puede realizar de diferentes maneras. Si usamos un único lenguaje de programación para la interfaz y la funcionalidad, lo usual es contar con un entorno de desarrollo integrado común para todas las fases de desarrollo como al utilizar la biblioteca swing con Java. Sin embargo, cuando el lenguaje final es uno y el lenguaje de la interfaz otro, como es el caso de las tecnologías basadas en XML que estamos viendo, normalmente se requiere de un proceso de **traducción** de los elementos XML a elementos que entienda la plataforma final, como en el caso de XAML que traduce cada elemento XML a clases de la plataforma .NET. En el caso de XUL es el motor de renderización el que se encarga de hacer las transformaciones necesarias para visualizar la interfaz. Este proceso puede ser más o menos automático en función de las tecnologías seleccionadas.

A continuación, veremos algunos ejemplos de herramientas existentes, tanto libres como propietarias, que difieren en la manera que tratan los archivos XML con las interfaces para incluirlos en la aplicación final.

### 2.5.1 Presentación de algunas herramientas

Las herramientas de diseño de interfaces gráficas de usuario forman parte de las herramientas RAD (Rapid Application Development, Desarrollo rápido de Aplicaciones).

El **principal objetivo** de estas herramientas es ocultar la sintaxis de los lenguajes de modelado y proporcionarles una interfaz que permita especificar adecuadamente el modelo de interfaz en los tres aspectos que hemos visto, a saber:

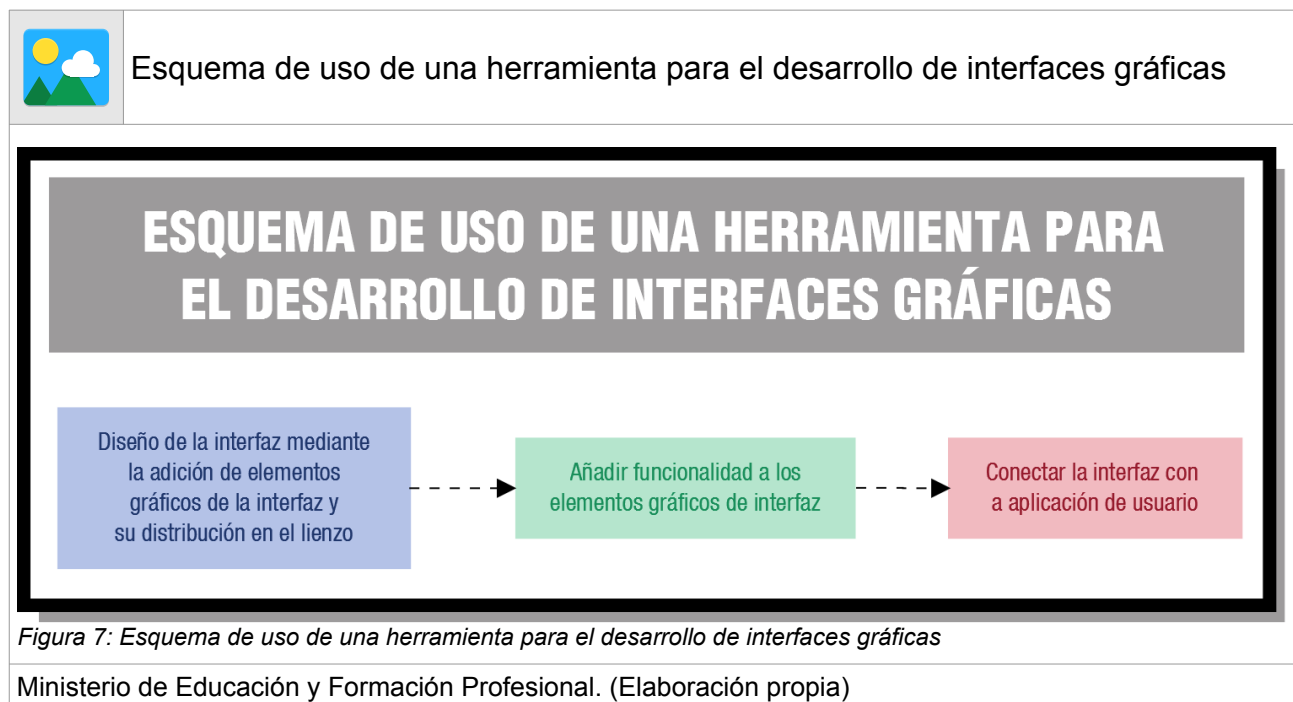
La interfaz se almacena en un archivo de texto plano siguiendo las directrices del estándar XML.

Estas herramientas, disponen de editores, intérpretes, generadores y otras aplicaciones útiles para llevar a cabo tareas relacionadas con la elaboración, manipulación o generación de modelos de interfaz.

Entre otras, con estas características podemos encontrar las siguientes aplicaciones:

- Libres:
  - QT Designer.
  - Glade.
  - Scene Builder
- Propietarias:
  - Expression Blend de Microsoft.

Las aplicaciones RIA (Rich Internet application), "aplicación de Internet enriquecida" o "aplicación rica de internet", es una aplicación web que tiene la mayoría de las características de las aplicaciones de escritorio.



## 2.6 Blend para Visual Studio

Blend para Visual Studio nos permite construir el diseño de aplicaciones para la Tienda de Windows, WPF y Silverlight. Actualmente se instala junto con Visual Studio. Hay que señalar que Blend no está disponible para Visual Studio Code.

Puedes descargar Visual Studio desde la página de Microsoft.





## Instalación de Visual Studio Community

Instalación en curso — Visual Studio Community 2022 — 17.3.6

Cargas de trabajo Componentes individuales Paquetes de idioma Ubicaciones de instalación

¿Necesita ayuda para elegir qué instalar? Más información

código fuente de Python.



### Desarrollo de Node.js

Compile aplicaciones de red escalables con Node.js, un entorno de ejecución JavaScript controlado por eventos...

### Móviles y de escritorio (5)



### Desarrollo de la interfaz de usuario de aplicaciones mu...

Cree aplicaciones de Android, iOS, Windows y Mac desde un único código base con C# mediante .NET MAUI.



### Desarrollo de escritorio de .NET

Compila WPF, Windows Forms y aplicaciones de consola mediante C#, Visual Basic y F# con .NET y .NET Framework...

### Detalles de la instalación

#### Desarrollo de escritorio de .NET

##### Incluido

- ✓ Herramientas de desarrollo de escritorio d...
- ✓ Herramientas de desarrollo de .NET Frame...
- ✓ C# y Visual Basic

##### Opcional

- ✓ Herramientas de desarrollo para .NET
- ✓ Herramientas de desarrollo de .NET Frame...
- ✓ Herramientas de Entity Framework 6
- ✓ Herramientas para generación de perfiles...
- ✓ IntelliCode
- ✓ Depurador Just-In-Time
- ✓ Live Share
- ✓ ML.NET Model Builder
- ✓ Blend for Visual Studio
- ☐ Compatibilidad con el lenguaje de escritor...
- ☐ PreEmptive Protection - Dotfuscator
- ☐ Herramientas de desarrollo de .NET Frame...
- ☐ Paquete de compatibilidad de bibliotecas...

### Ubicación

C:\Program Files\Microsoft Visual Studio\2022\Community [Cambiar...](#)

Si continúa, indica que acepta la [licencia](#) de la edición de Visual Studio que ha seleccionado. También puede descargar otro software con Visual Studio. Este software tiene una licencia aparte, como se explica en los [avisos de terceros](#) o en la licencia que incluye el software. Si continúa, indica que también acepta esas licencias.

Espacio total necesario 6,86 GB

Instalar durante la descarga

Instalar

Figura 8: Instalación de Visual Studio Community



## Pantalla de acceso a Blend Studio

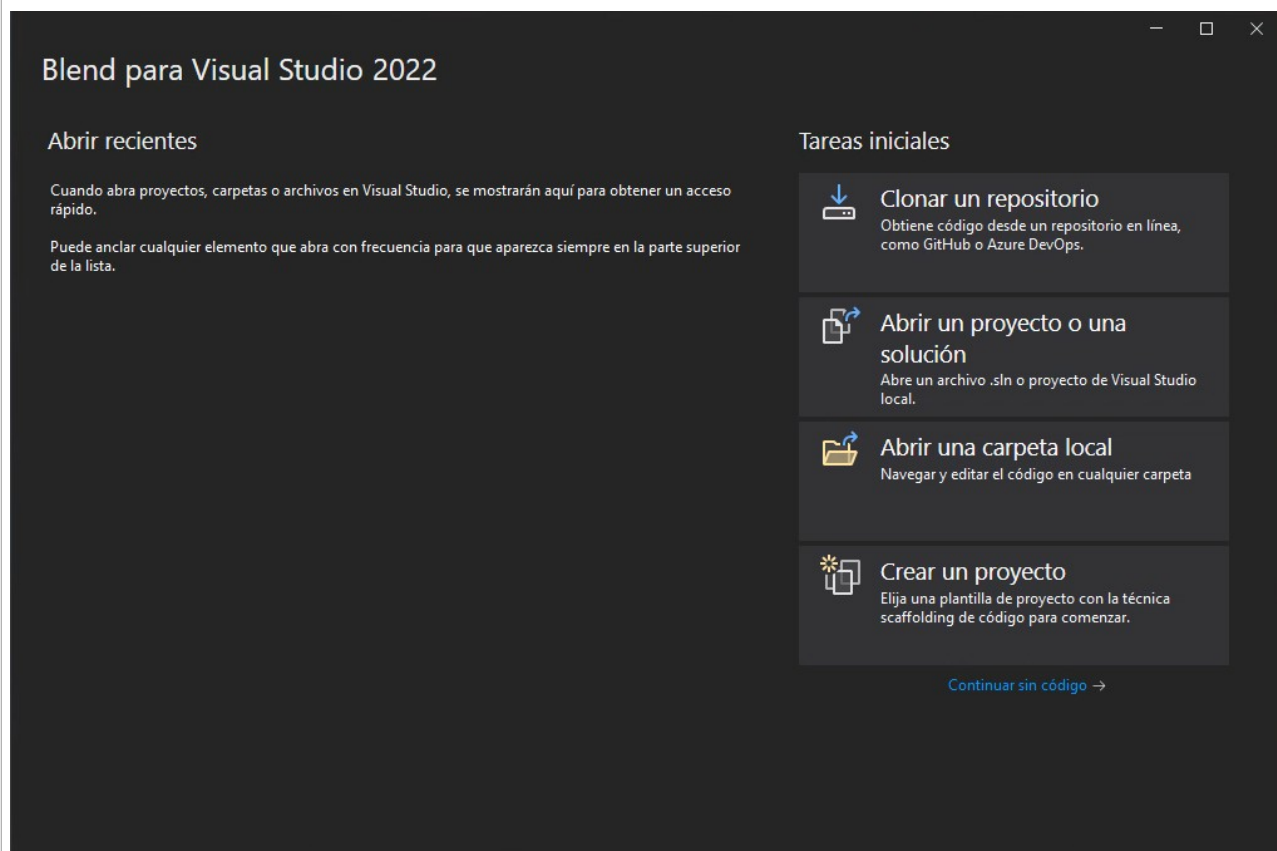


Figura 9: Pantalla de acceso a Blend Studio

Montaña Martín Vergel (Elaboración propia)

Para comenzar a trabajar, necesitaremos crear un proyecto. Para ello, podemos utilizar plantillas existentes que nos facilitan el diseño de la aplicación.



## Entorno de trabajo de Blend Studio

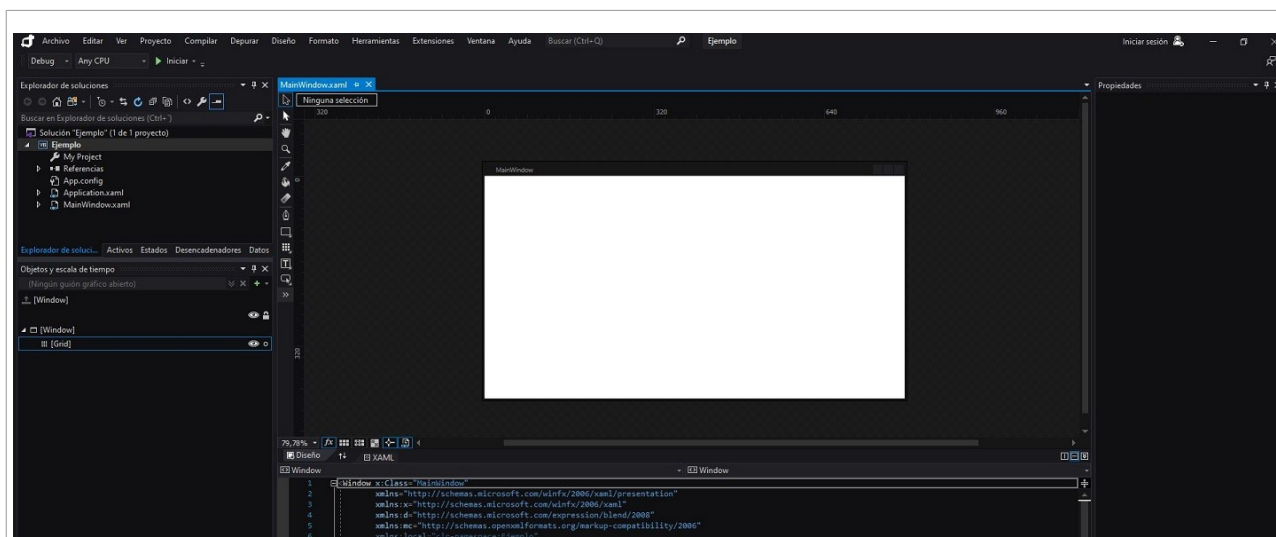


Figura 10: Entorno de trabajo de Blend Studio

Montaña Martín Vergel (Elaboración propia)

## 2.6.1 Controles. Contenedores

Los controles que podemos utilizar los podemos encontrar en la Paleta. Se muestra en el lado izquierdo y organiza los widgets por categorías. Las categorías que podemos encontrar son:

Proyecto, Controles, Estilos, Controles de componentes, Formas, Efectos, Multimedia, Categorías y Ubicaciones.



Entorno de trabajo de Blend Studio

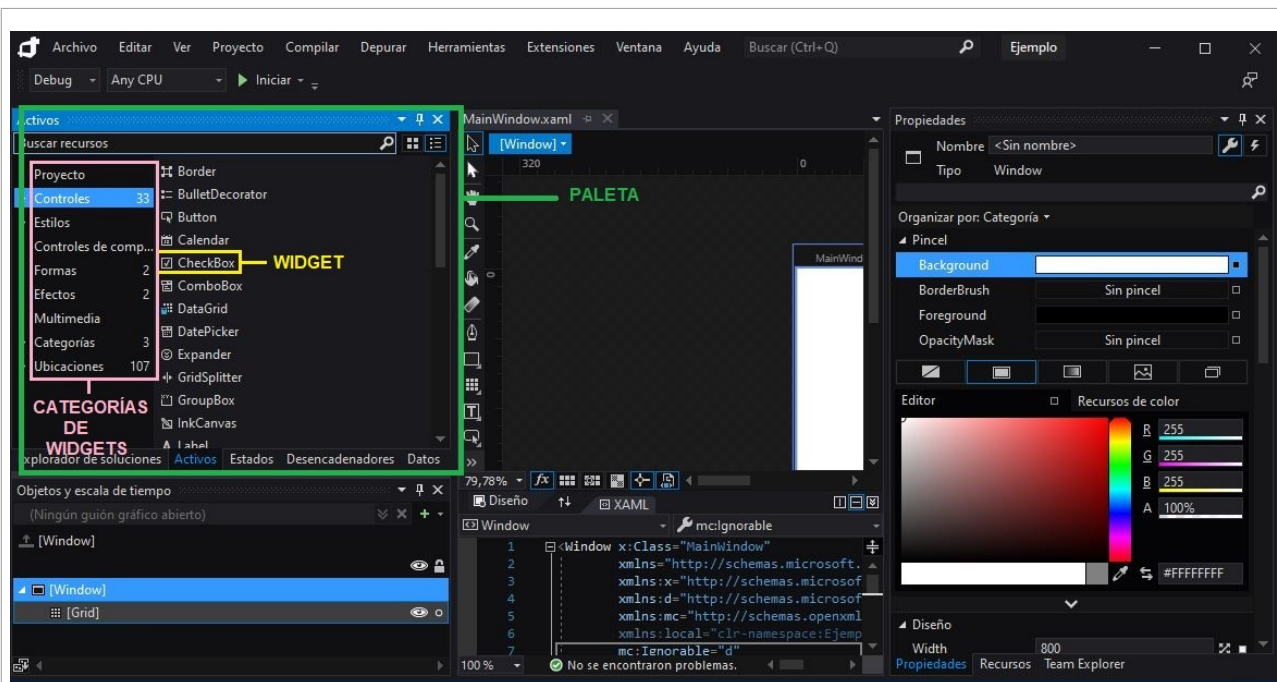


Figura 11: Entorno de trabajo de Blend Studio

Montaña Martín Vergel (Elaboración propia)

Los widgets contenedores están situados en la paleta dentro del menú Controles -> Paneles.

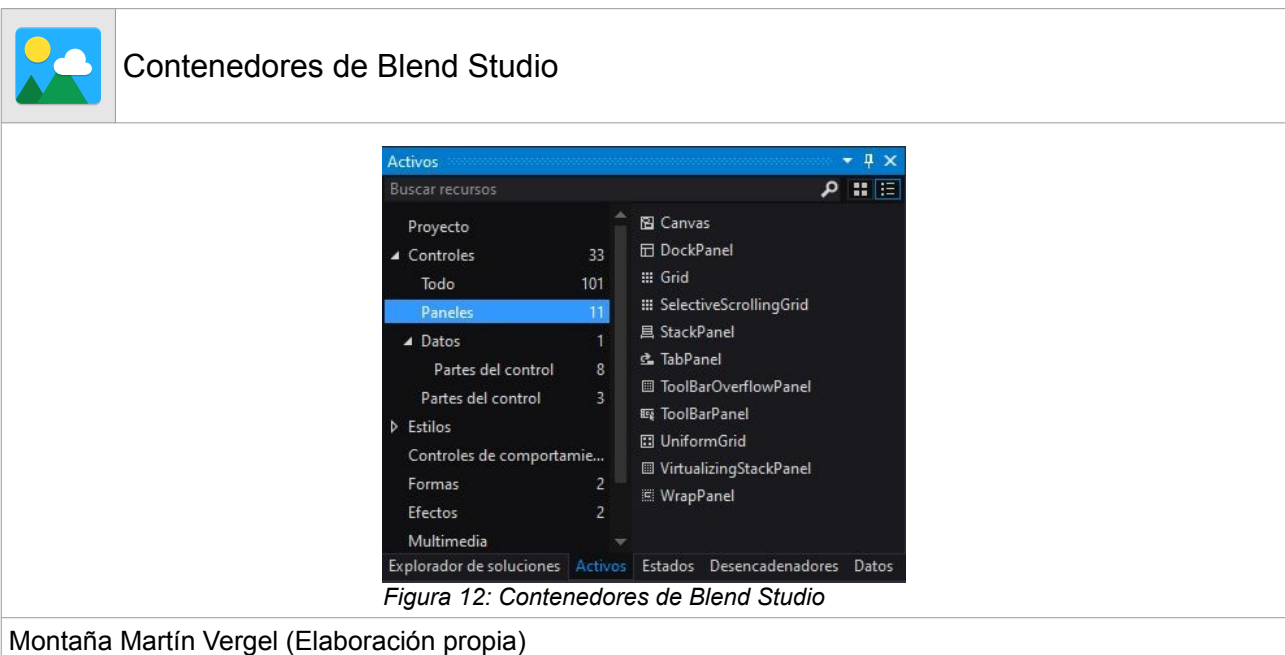


Figura 12: Contenedores de Blend Studio

Montaña Martín Vergel (Elaboración propia)

## 2.6.2 Controles y propiedades

Los controles más utilizados en Blend Studio son: Cuadros de textos o entrada de texto, etiquetas, botones, opciones, listas despegables y casillas de verificación.



## Botón en Blend Studio

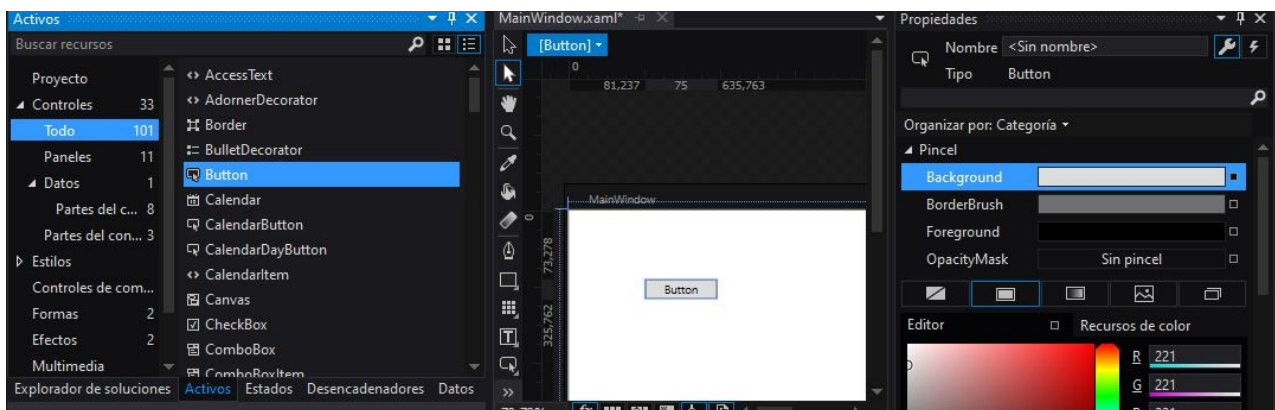


Figura 13: Botón en Blend Studio

Montaña Martín Vergel (Elaboración propia)



## Etiqueta en Blend Studio

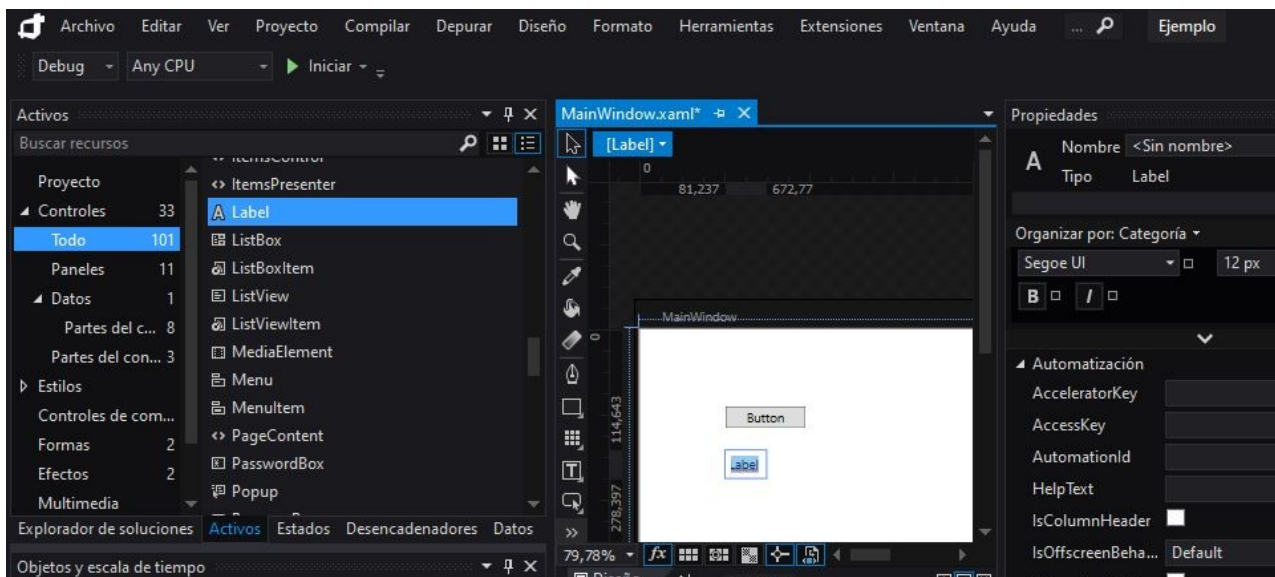


Figura 14: Etiqueta en Blend Studio

Montaña Martín Vergel (Elaboración propia)

- Entrada de texto (Text Entry o TextBox): nos permiten introducir datos en nuestras aplicaciones, aunque también es posible mostrar información a los usuarios en tiempo de ejecución a través de ellos.

- Etiqueta (Label): sirven para mostrar por pantalla texto.
- Botón (Button): nos permite que la aplicación ejecute un conjunto de acciones cuando el usuario los seleccione.
- Opciones (Radio / RadioButton): permite indicar varias opciones para que el usuario solo seleccione una.
- Casillas de verificación: permite presentar una o varias opciones. El usuario podrá seleccionar una o varias con independencia de la que seleccionemos.
- Lista desplegable: nos permite mostrar un conjunto de valores para que el usuario seleccione uno de ellos.

La mayoría de los controles, tienen una serie de propiedades: el color, la ubicación, los estilos que se le aplican, el tamaño, etc.



## Propiedades de los controles en Blend Studio

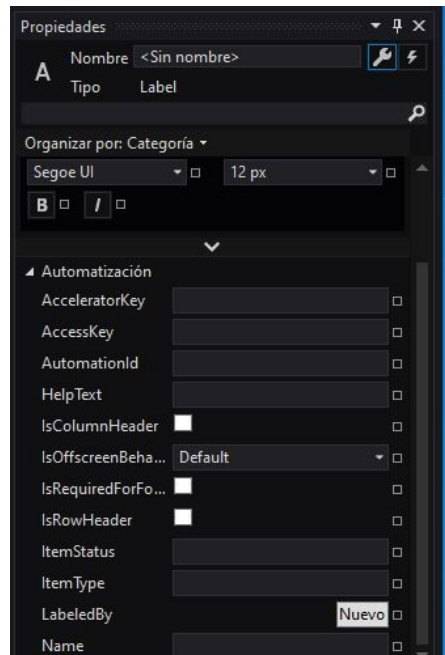


Figura 15: Propiedades de los controles en Blend Studio

Montaña Martín Vergel (Elaboración propia)

## 2.7 Glade

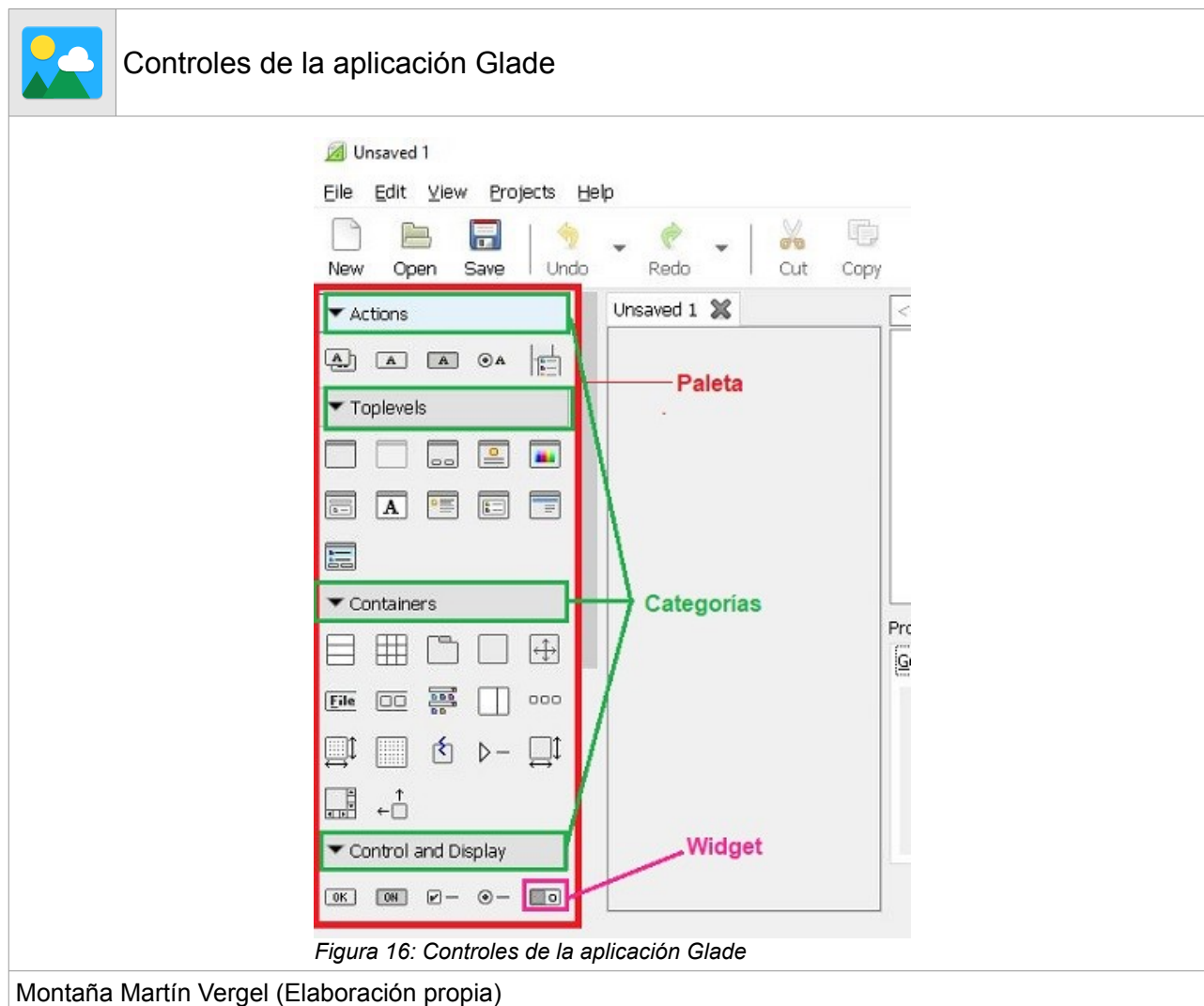
Glade (o Glade Interface Designer) Diseñador de interfaces Glade, es una herramienta de desarrollo de interfaces gráficas mediante GTK/GNOME. Es independiente del lenguaje de programación y no genera código fuente sino un archivo XML.

Las herramientas de Glade son implementadas como widgets, Esto permite que se puedan integrar más fácilmente en los IDE's.

Glade utiliza un tipo de formato XML denominado GtkBuilder para almacenar los elementos de las interfaces diseñadas. Estos archivos pueden emplearse para construir la interfaz en tiempo de ejecución.

### 2.7.1 Paletas y vistas

Glade utiliza Widgets o controles para construir el diseño de la interfaz. Estos se encuentran situados en la paleta. Al acceder a la aplicación, la paleta se encuentra situada en el lado izquierdo y contiene los Widgets organizados por categorías. Las categorías son: Actions, TopLevels, Containers, Color and Display, Composite Widgets y Miscellaneous.



### 2.7.2 Contenedores de controles

Para organizar los controles o Widget dentro del proyecto podemos utilizar los componentes denominados cajas o contenedores (Containers). En la siguiente imagen se puede apreciar cuales son:





## Contenedores de la aplicación Glade.

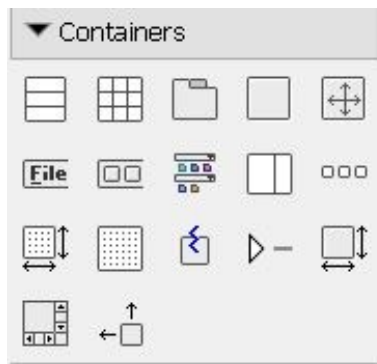


Figura 17: Contenedores de la aplicación Glade.

Montaña Martín Vergel (Elaboración propia)

Al utilizar los contenedores horizontales y verticales, Glade te pregunta cuántas filas o columnas queremos añadir. Este número posteriormente puede ser modificado ya que nos permite eliminar filas o columnas e insertar filas y columnas.

Una vez añadido todos los contenedores que necesitemos, podemos añadir los widgets que vayamos a utilizar en su interior. Estos elementos pueden ser cortados, copiados, pegados o borrados desde el menú Edit o seleccionando el botón derecho del ratón sobre ellos.

Cada Widget tendrá un nombre único identificativo dentro del proyecto.

### 2.7.3 Controles y propiedades. Alineación, tamaño y ubicación de los controles

La mayoría de los controles poseen una serie de propiedades que pueden ser modificadas. Por ejemplo, el tamaño, el color, la posición, etc...

Los controles que más se utilizan son:

- **Entrada de texto (Text Entry o TextBox):** nos permite introducir información a la aplicación, aunque también es posible utilizarlo para mostrar información a través de ellos.
- **Etiqueta (Label):** nos permite modificar texto a través de ella y en tiempo de ejecución no es posible ser modificada.
- **Botón (Button):** nos permite insertar un botón. Tienen la característica de que se pueden insertar un texto indicando la acción que se realizará al presionarla.
- **Opciones (Radio / RadioButton):** nos permite seleccionar un valor entre posibles valores.
- **Casillas de verificación (Check / CheckBox):** Se utilizan para representar valores que pueden ser o no seleccionados por los usuarios. Ejemplo típico: la

casilla que se muestra cuando tenemos que aceptar las condiciones de la licencia de una aplicación en su proceso de instalación.

- **Lista Desplegable (Combo / Select / Combox):** típica lista que nos presenta un conjunto de valores para seleccionar uno. Por ejemplo, podemos mostrar un desplegable con las provincias españolas para que el usuario seleccione en la que resida.

Al seleccionar el objeto en la parte de la derecha nos aparece el panel de propiedades (Properties):

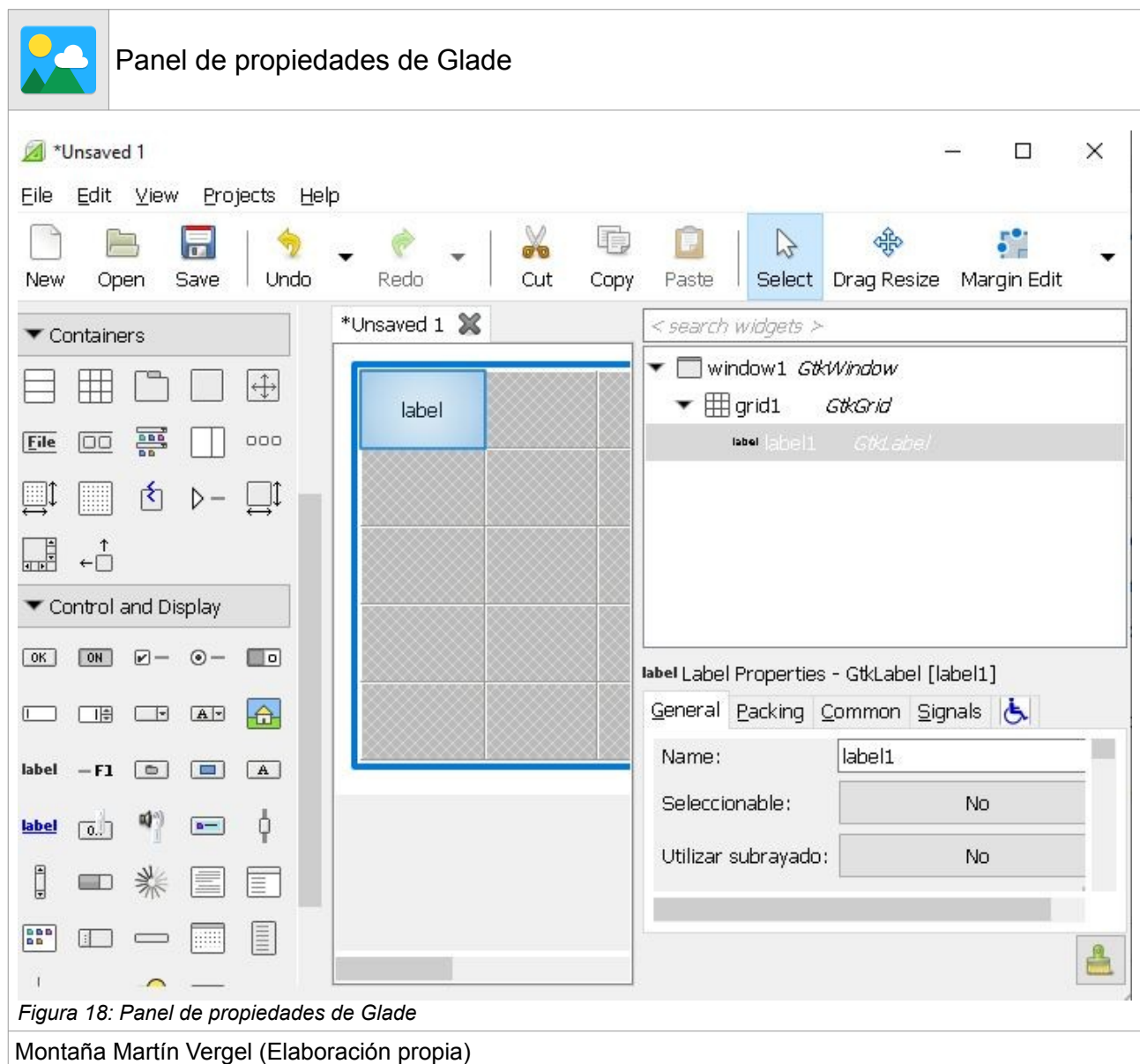


Figura 18: Panel de propiedades de Glade

Montaña Martín Vergel (Elaboración propia)

Podemos colocar los controles en cualquier lugar de la interfaz de la misma forma que podemos variar su tamaño y posición.

En Glade, para colocar un control solo hay que arrastrarlo hacia el lugar del contenedor en donde queramos colocarlo. Por lo tanto, el tamaño, la posición y la alineación del control estará condicionado a la distribución de los contenedores.

## 2.8 Scene Builder

La aplicación Scene Builder pertenece a la compañía Gluon. Se trata de una aplicación que nos permite construir la interfaz de una aplicación independientemente del código de programación. El código asociado a la creación de la interfaz se almacena en un fichero XML con el formato FXML,

Es una aplicación gratuita y de código abierto. Existen versiones para los sistemas operativos Windows, Linux y Mac OS.

Se suele utilizar junto con aplicaciones JavaFX y se puede integrar en IDE's como Eclipse o NetBeans.

Se utiliza de forma fácil ya que permite arrastrar y soltar los elementos que queremos que aparezcan en las interfaces.

Para utilizarlo es necesario descargarlo e instalarlo. Lo podemos hacer desde el siguiente enlace: [Descargar Scene Builder](#).

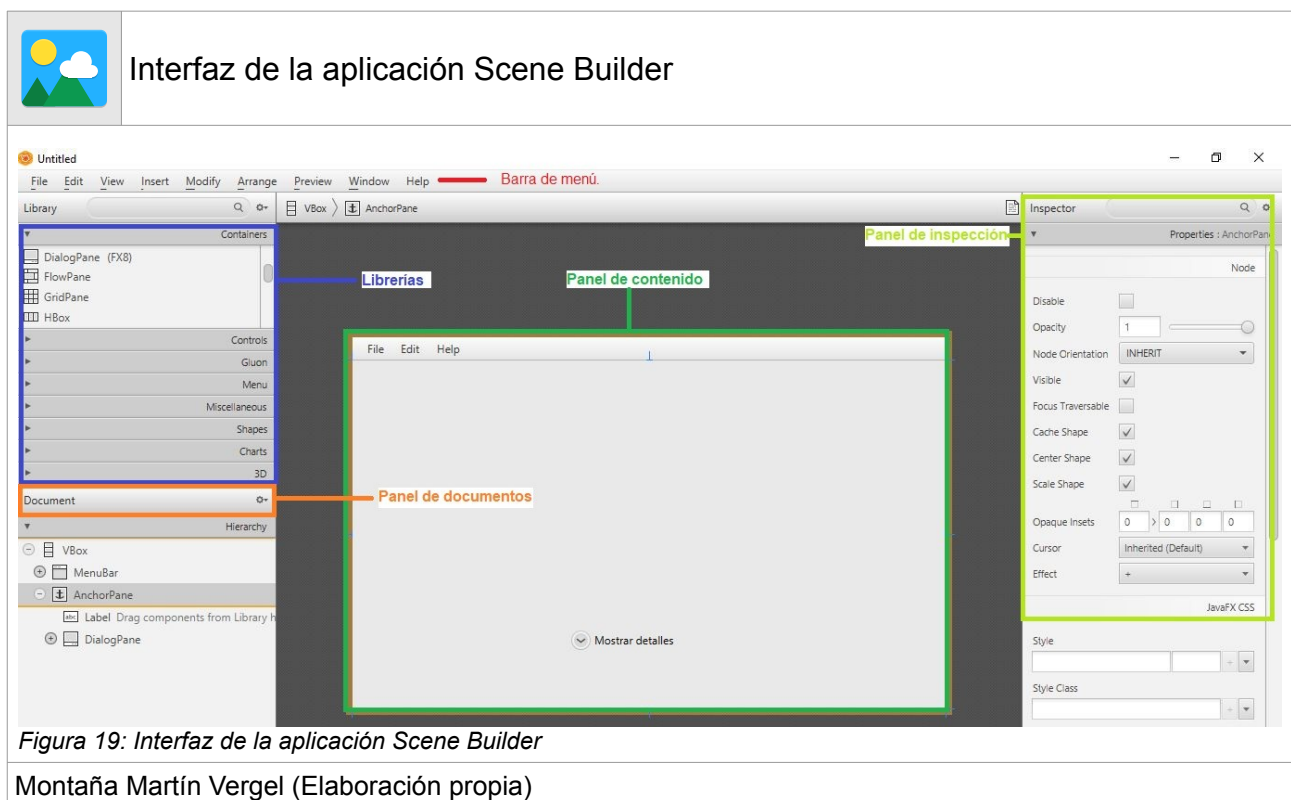


Figura 19: Interfaz de la aplicación Scene Builder

Montaña Martín Vergel (Elaboración propia)

Dentro de las librerías que nos aparecen en el panel situado a la izquierda tenemos las siguientes categorías: containers, controls, Gluon, Menú, Miscellaneous, shapes, Charts y 3D.

- **Containers:** el primer elemento que introduciremos para poder añadir el resto de elementos sobre el escenario.
- **Controls:** agrupa los controles como botones, casillas de verificación, desplegables, etc.

- **Gluon:** agrupan controles personalizados por la empresa Gluon y que reutilizan en las aplicaciones de la compañía.
- **Menú:** agrupa los controles como barra de menús,
- **Miscellaneous:** agrupa diversos controles: escenario, canvas, tooltip, etc.
- **Shapes:** nos permite utilizar herramientas de dibujo con círculos, textos, polígonos, líneas, etc.
- **Charts:** agrupa controles para añadir gráficos.
- **3D:** herramientas para modificar las perspectivas de los objetos, luminosidad, etc.

## 2.9 Eventos. Secuencia de eventos

En la programación dirigida por eventos, tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema. En este tipo de programas serás el usuario quien determine el flujo de la aplicación en función de lo que vaya accionando en el programa.

El programador debe de definir los eventos que manejarán sus aplicaciones y la acción que se realizará al producirse cada uno de ellos, lo que se conoce como **administrador de eventos**.

Cuando ejecutamos una aplicación dirigida por eventos, al comienzo de ella, se producirán las inicializaciones correspondientes y a continuación el programa quedará inactivo hasta que se produzca un evento.

La programación dirigida por eventos es la base de lo que se denomina interfaz de usuario.

Los pasos para la construcción de una aplicación dirigida a eventos son:

- 1-. Escribir una serie de subrutinas llamados rutinas controlador de eventos que se encargarán de controlar los eventos a los que el programa principal responderá. Actualmente, muchos entornos de programación suministran plantillas de eventos para que el programador solo tenga que introducir su código asociado.
- 2-. Se debe de enlazar los controladores de eventos a los eventos para que se ejecute el código cuando se genere el evento.
- 3-. Debemos de definir un bucle principal. Este bucle se encargará de comprobar la ocurrencia de los hechos y a continuación llamará al controlador de evento correspondiente para procesarlo. Actualmente los entornos de programación facilitan la creación de este bucle. Esto facilita el trabajo al programador.

## 2.10 Análisis y edición de documentos XML

Siempre que trabajemos con documentos XML es conveniente su validación. La validación consiste en la comprobación de que un documento XML se encuentre bien

formado y se ajusta a una estructura bien definida. Un documento bien formado sigue las reglas básicas de XML establecidas para el diseño de documentos. Un documento válido, además, respecta las normas indicadas en el fichero DTD (definición tipo de documento) al que se debe de encontrar asociado.

La validación consiste en:

- 1-. La corrección de los datos: permite detectar valores nulos o fuera de rangos.
- 2-. La integridad de los datos: se comprueba que toda la información obligatoria se encuentre presente en el interior del documento.
- 3-. El entendimiento compartido de los datos: se comprueba que el emisor y el receptor van a recibir el documento de la misma forma y que lo puedan interpretar igual.

Para la creación de documentos XML existen herramientas que evitan que se introduzcan datos erróneos, errores tipográficos, sintácticos y de contenidos. Estas herramientas se conocen con el nombre de Editores XML.

Los editores de XML tienen que ser capaces de:

- 1-. Leer la DTD correspondiente al documento y presentar una lista desplegable con los elementos disponibles enumerados en la DTD, evitando la inclusión de algún elemento incluido en el esquema.
- 2-. Advertir del olvido de una etiqueta obligatoria e incluso no permitir este tipo de descuido o errores, No permite la finalización del documento si existen errores.

## **2.11 Ejemplo de desarrollo de una interfaz utilizando XML**

Scene Builder es una aplicación de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario sin codificación. Los usuarios pueden arrastrar y soltar componentes de la IU a un área de trabajo, modificar sus propiedades, aplicar hojas de estilo y el código que genera lo almacena en formato FXML. El resultado es un archivo FXML que luego se puede combinar con un proyecto Java al vincular la interfaz de usuario a la lógica de la aplicación.

Para utilizar Scene Builder es necesario tenerlo instalado en el equipo.

Para instalarlo en Linux es necesario acceder a la web [GluonHQ](https://gluonhq.com) y descargar el paquete de instalación adecuado para la distribución. Por ejemplo, para instalarlo en Ubuntu hay que descargar el instalador “Linux Deb”. Si el sistema operativo tiene soporte para Flatpak se puede instalar de forma directa desde el gestor de software Flatpak.

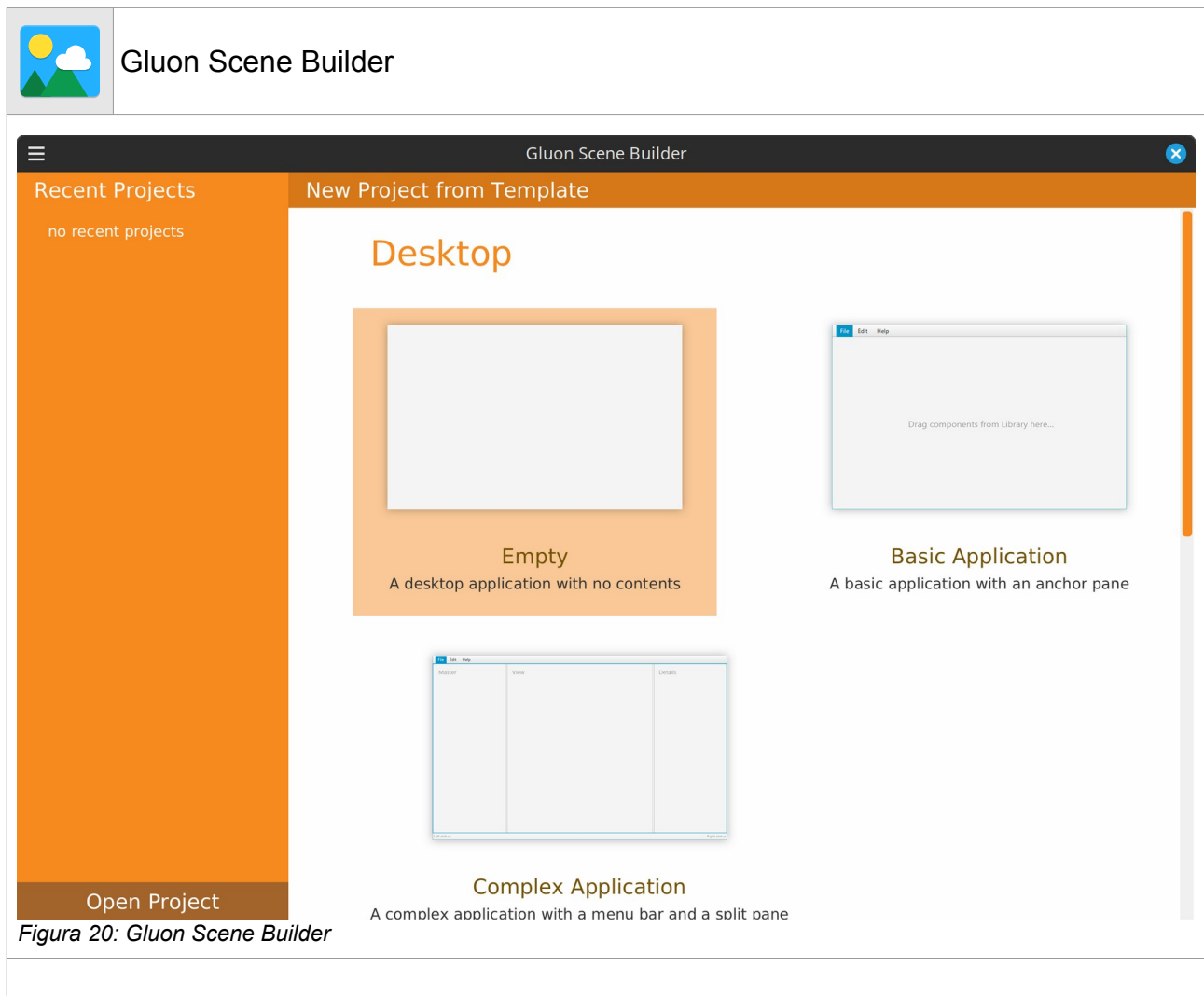


Figura 20: Gluon Scene Builder

Para modificar los ficheros FXML a través de la aplicación Scene Builder desde NetBeans, es necesario modificar la configuración de NetBeans. Para ello, accedemos a la opción Herramientas → Opciones y seleccionar la pestaña JavaFX. En el apartado Scene Builder Home deberemos de indicar la ruta absoluta en donde se encuentre instalada la aplicación Scene Builder.



## Configuración en NetBeans de JavaFX

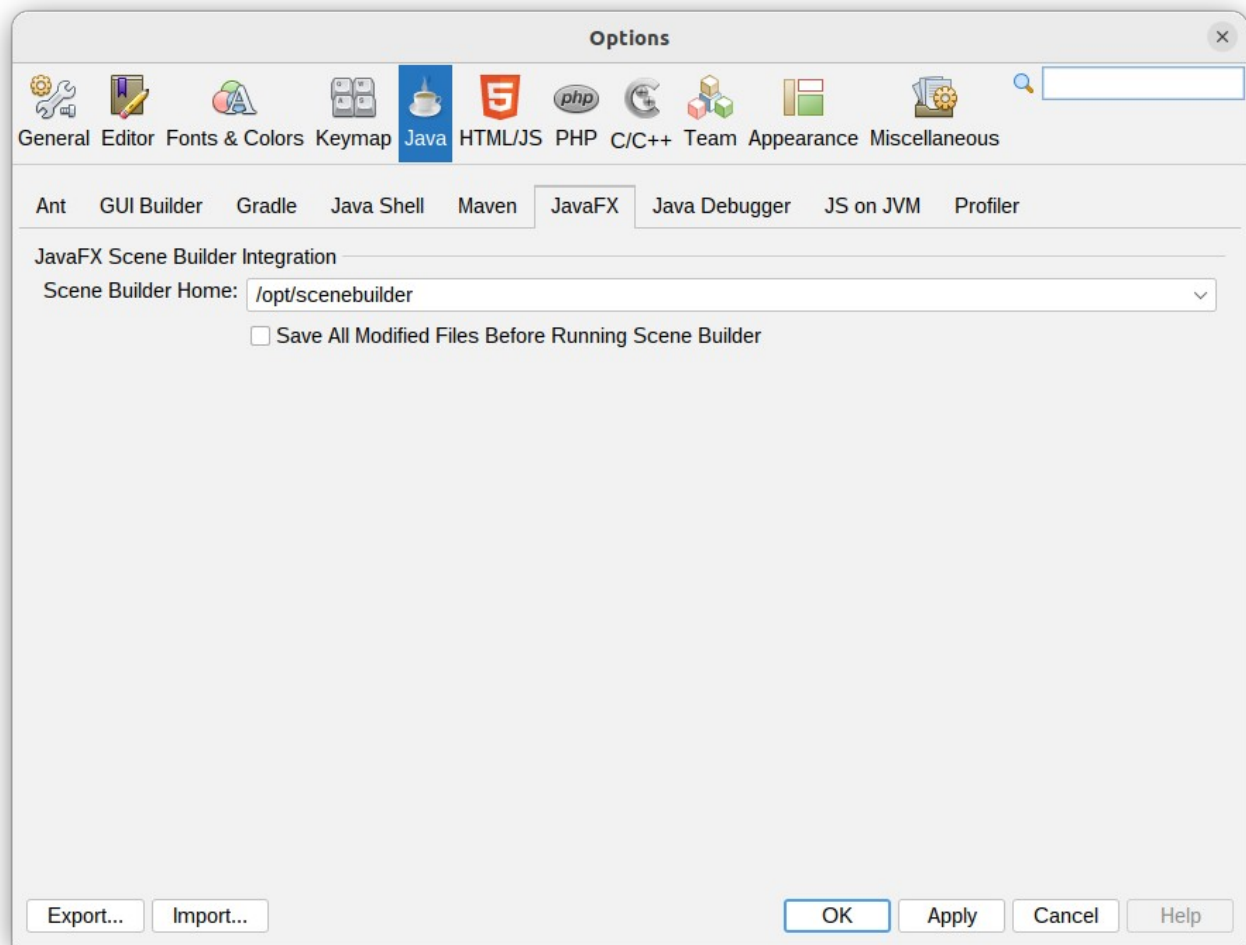


Figura 21: Configuración en NetBeans de JavaFX

Una vez seleccionado pulsaremos Aplicar y Aceptar. Una vez hecho esto, ya estamos en disposición de crear proyectos JavaFX desde NetBeans, utilizando la aplicación Scene Builder para crear la interfaz de la aplicación.

### 2.11.1 Descripción del problema

En primer lugar, hay que destacar que en estos materiales se pretende dar un vistazo rápido al funcionamiento de una herramienta para la creación de interfaces de usuario usando XML, así como la posterior integración de esta interfaz en una aplicación. Por lo tanto, no podrá considerarse esta unidad como un manual completo de aprendizaje de JavaFX Scene Builder, sino como ejemplo didáctico de la materia que se pretende trasladar al alumnado.

Como caso de uso se implementará un gestor de tareas.





El proyecto que se creará será Java with Maven

### 2.11.2 Creación proyecto JavaFX

Se comenzará por crear en NetBeans un nuevo proyecto JavaFX. Para ello, se seleccionará Archivo → Nuevo proyecto → Java with Maven → FXML JavaFX Maven Archetype (Gluon):



Creación del proyecto Java FX

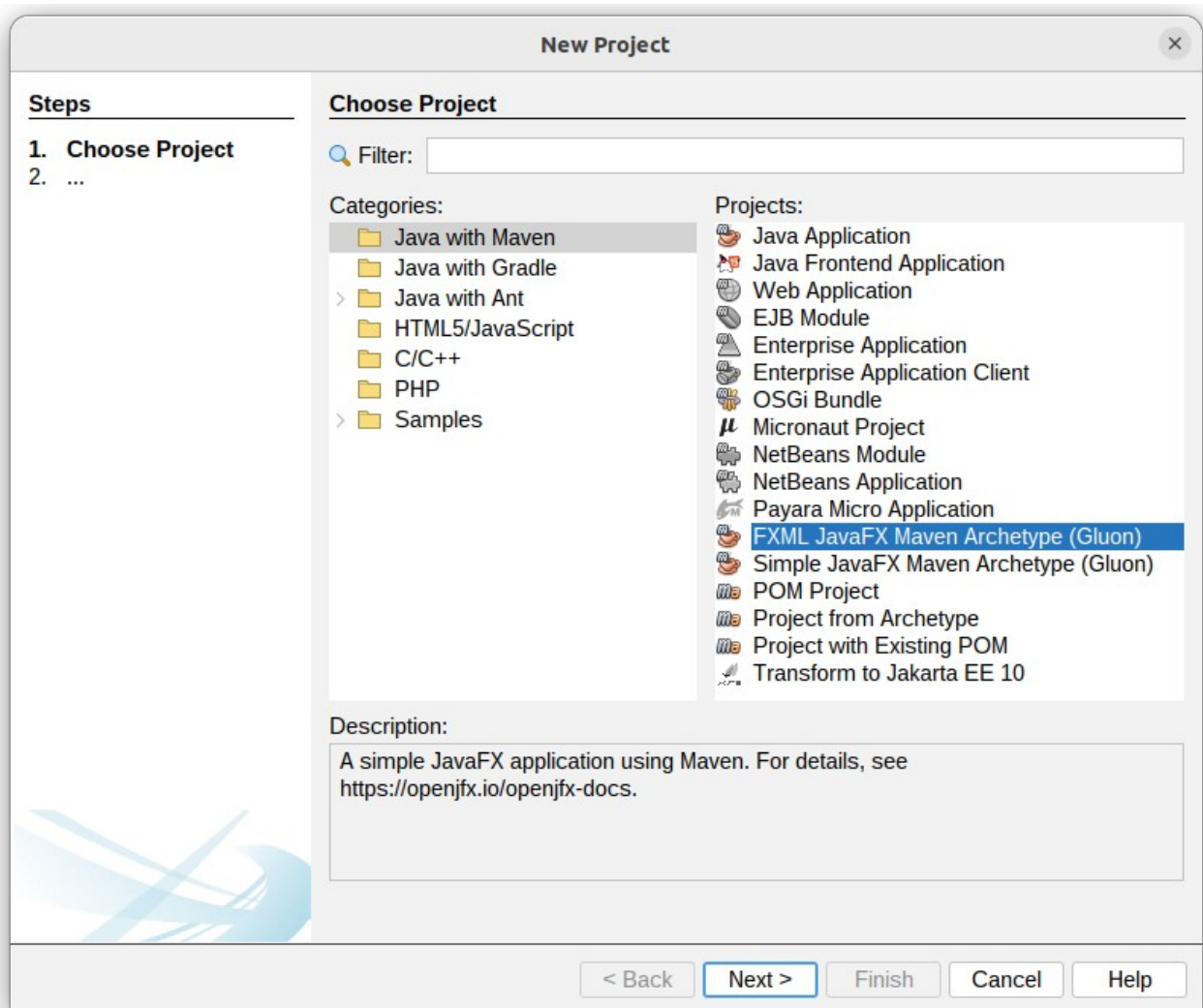


Figura 22: Creación del proyecto Java FX



A continuación, se introduce el nombre del proyecto, por ejemplo, TaskManager y se pulsa Terminar.

Si se observan los ficheros que se han creado al generar el proyecto, se puede observar que se han creado tres ficheros entre los que destaca el fichero con extensión fxml. Si lo observamos podemos ver que en su interior aparece código XML. A través del código XML podremos configurar la interfaz de la aplicación. Por defecto, al generar una nueva aplicación JavaFX, se inserta en la aplicación un contenedor para albergar un botón que contiene la etiqueta o el texto Púlsame.

En la siguiente imagen puede verse el proyecto generado y los ficheros que contiene:

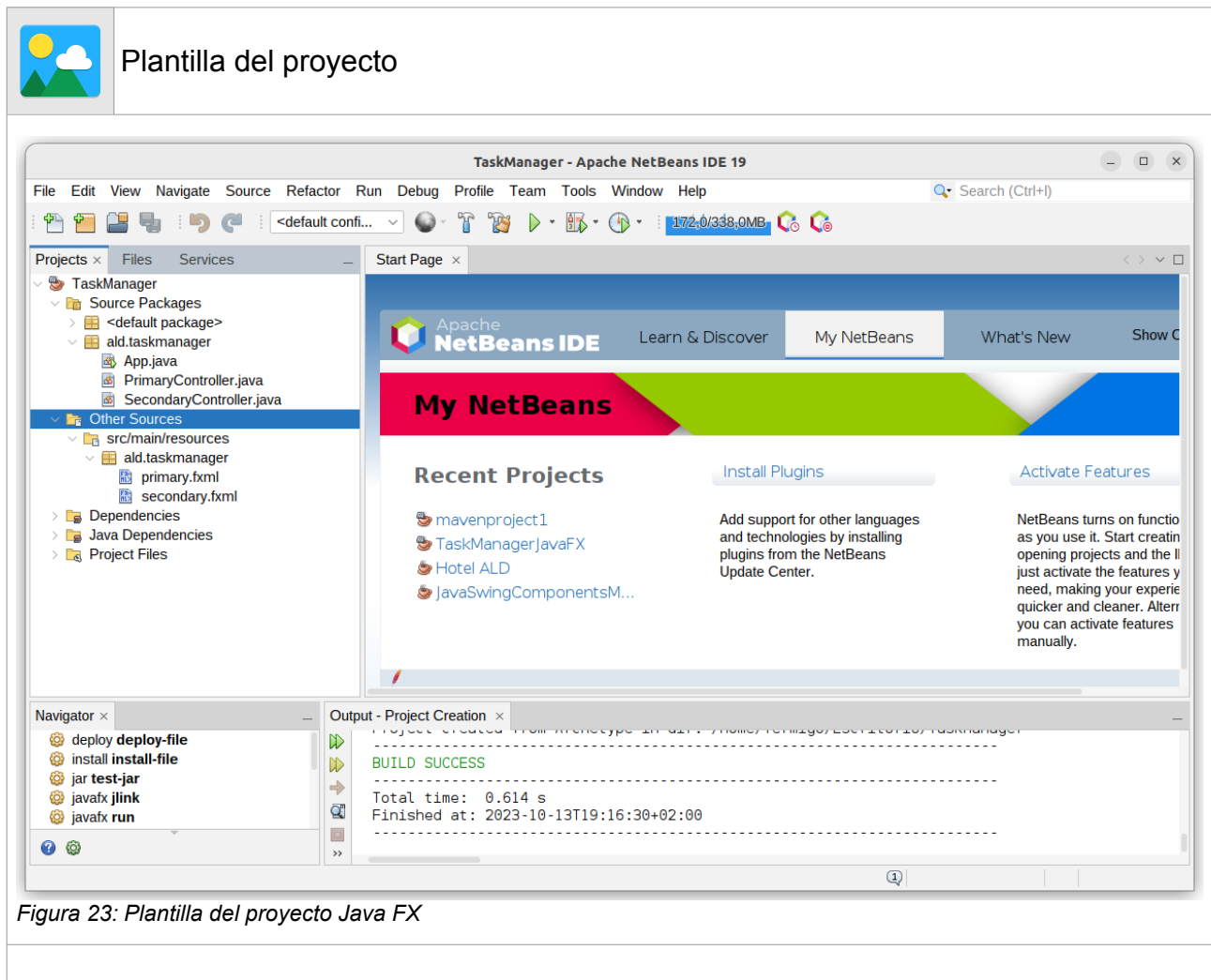


Figura 23: Plantilla del proyecto Java FX

Si se ejecuta la aplicación, se mostrará el formulario junto con el botón indicado.



*Figura 24: Vista de la aplicación de ejemplo*

### 2.11.3 Edición de la interfaz

Para comenzar el diseño de la interfaz, se seleccionará el fichero en formato fxml, en el ejemplo tiene el nombre primary.xml. Accederemos al menú contextual y seleccionamos la opción Open. Se verá cómo se comienza a ejecutar la aplicación Scene Builder y en su interior aparece la configuración que hasta ahora tiene el FXML.



## Edición del fichero FXMLDocument dentro de la aplicación Scene Builder

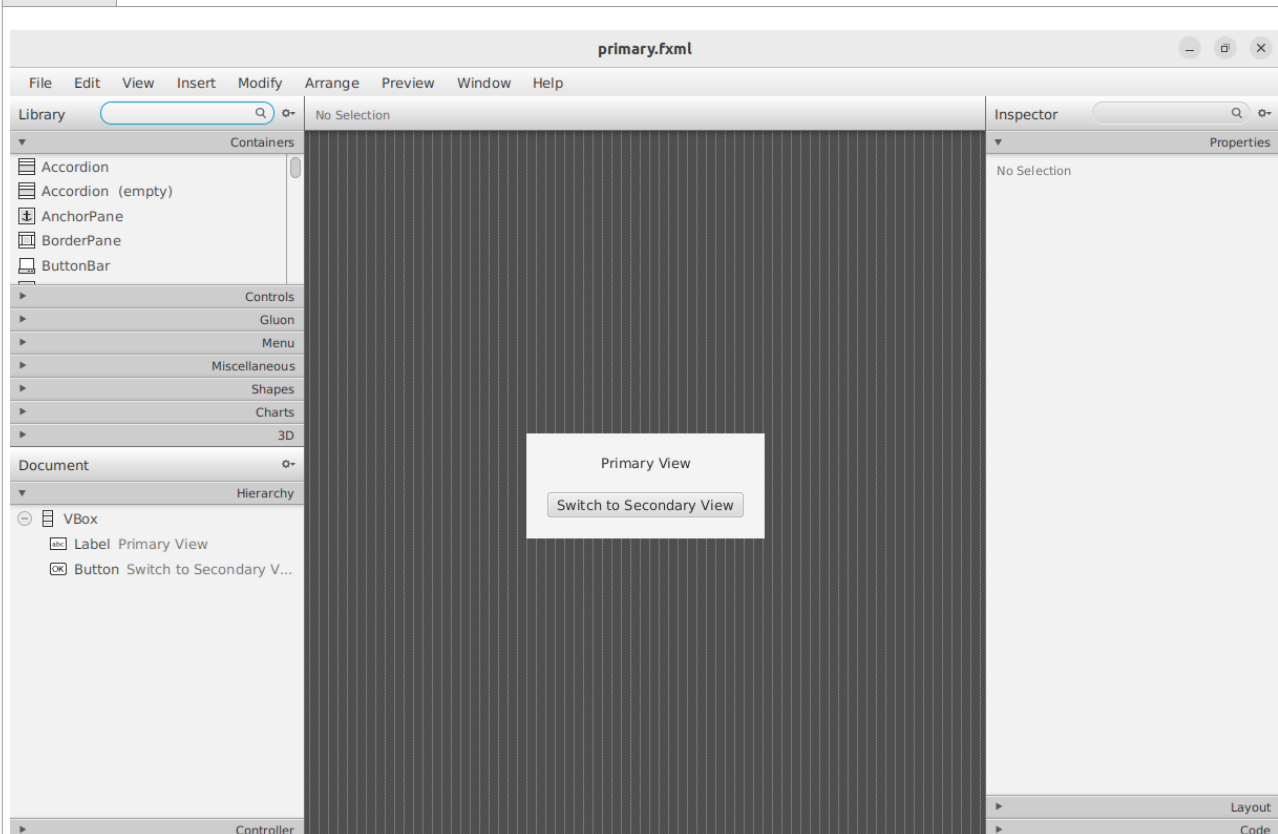


Figura 25: Edición del fichero FXMLDocument dentro de la aplicación Scene Builder

### Requisitos de la interfaz

Se pretende crear una interfaz para gestionar una lista de tareas del siguiente modo:

- Menú con la opción Salir
- Descripción de la tarea (TextField)
- Lista de tareas pendientes (ListView)
- Botón para añadir la tarea (Button)

En NetBeans se eliminarán los archivos correspondientes a la vista secundaria y el controlador secundario. La vista principal se renombrará como TaskManagerDialog y el controlador principal se renombrará como TaskManagerController.

En Scene Builder se eliminarán todos los componentes gráficos de la interfaz y se añadirá un AnchorPane, al que se le añadirán los elementos del siguiente modo:



## Creación de la vista del proyecto

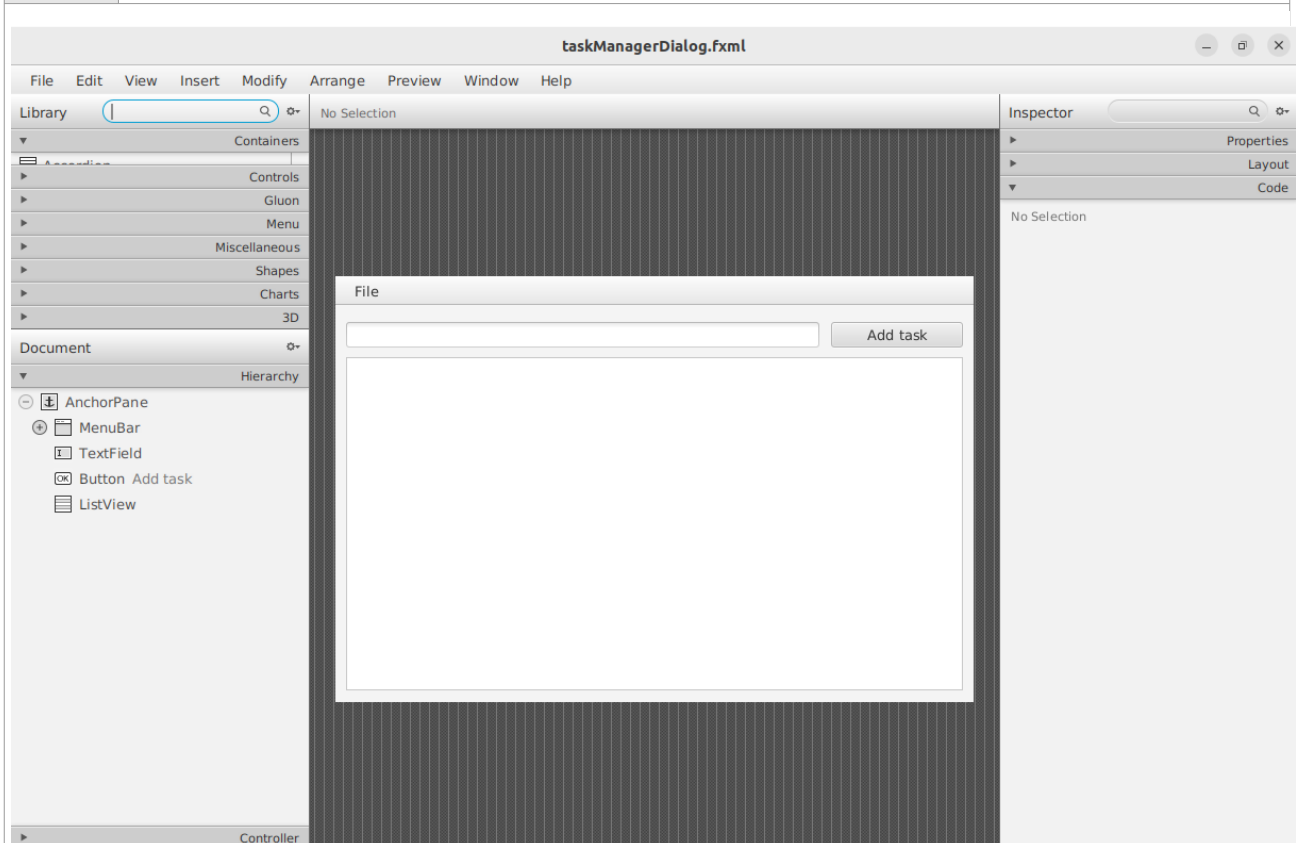


Figura 26: Creación de la vista del proyecto

En el botón Add Task hay que configurar fx:id y On Action:



## Configuración de elementos

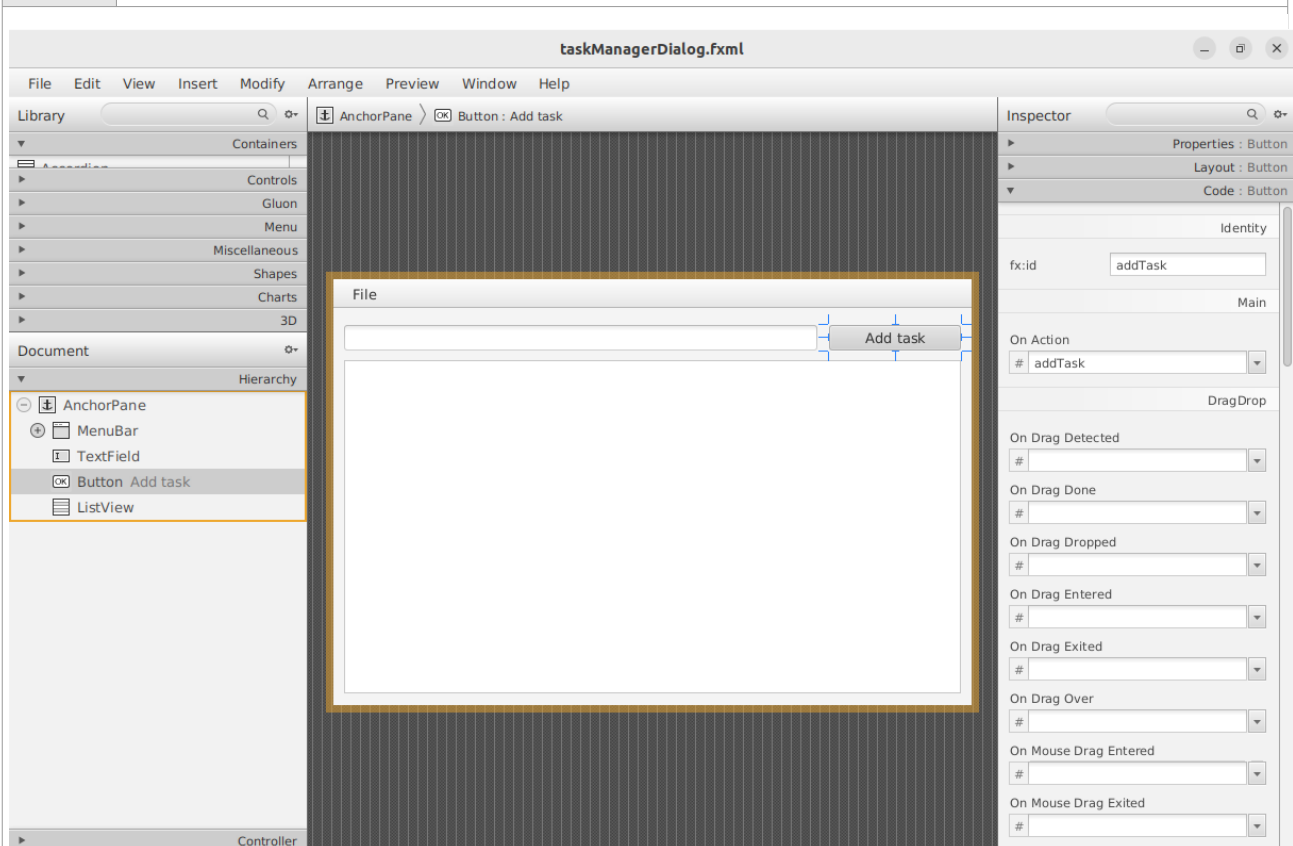


Figura 27: Configuración de elementos

Hay que señalar que es necesario guardar los cambios efectuados.

### 2.11.4 Implementación del código fuente

En NetBeans, el código App.java es el siguiente:



## Código fuente

```
package ald.taskmanager;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

/**
 * JavaFX App
 */
public class App extends Application {
```

```

private static Scene scene;

@Override
public void start(Stage stage) throws IOException {
    scene = new Scene(loadFXML("taskManagerDialog"), 640, 480);
    stage.setScene(scene);
    stage.show();
}

static void setRoot(String fxml) throws IOException {
    scene.setRoot(loadFXML(fxml));
}

private static Parent loadFXML(String fxml) throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
    return fxmlLoader.load();
}

public static void main(String[] args) {
    launch();
}
}

```

El código TaskManagerController.java es el siguiente:



### Código fuente

```

package ald.taskmanager;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.ListView;
import javafx.scene.control.MenuItem;
import javafx.scene.control.TextField;

public class TaskManagerController {

    @FXML
    private MenuItem quit;
    @FXML
    private TextField taskDescription;
    @FXML
    private Button addTask;
    @FXML
    private ListView<String> taskList;

    @FXML
    private void quit(ActionEvent event) {
        System.exit(0);
    }

    @FXML
    private void addTask(ActionEvent event) {
        if (!this.taskDescription.getText().isEmpty()) {
            this.taskList.getItems().add(this.taskDescription.getText());
        }
    }
}

```