

Módulo 7

Desarrollo de Interfaces

```
function updatePhotoDescription() {  
    if (descriptions.length > (page * 9) + (currentImageSubstring() - 1)) {  
        document.getElementById('bigImageDesc').innerHTML = descriptions[page * 9 + (currentImageSubstring() - 1)];  
    }  
}  
  
function updateAllImages() {  
    var i = 1;  
    while (i < 10) {  
        var elementId = 'foto' + i;  
        var elementIdBig = 'bigImage' + i;  
        if (page * 9 + i - 1 < photos.length) {  
            document.getElementById(elementId).src = 'images/min/' + photos[page * 9 + i - 1];  
            document.getElementById(elementIdBig).src = 'images/max/' + photos[page * 9 + i - 1];  
        } else {  
            document.getElementById(elementId).src = 'images/min/default.jpg';  
            document.getElementById(elementIdBig).src = 'images/max/default.jpg';  
        }  
        i++;  
    }  
}
```

UF1. DISEÑO E IMPLEMENTACIÓN DE INTERFACES.....	4
1. Confección de interfaces de usuario.	4
1.1. Librerías de componentes disponibles para los distintos sistemas operativos y lenguajes de programación. Características.	9
1.2. Componentes: características y campos de aplicación.	12
1.3. Eventos, escuchadores y acciones a eventos.	13
1.4. Edición del código generado por las herramientas de diseño.	13
1.5. Clases, propiedades, métodos.....	16
2. Generación de interfaces a partir de documentos XML:	20
2.1. Lenguajes de descripción de interfaces basadas en XML. Ámbito de aplicación.	20
2.2. XAML (Extensible Application Markup Language).....	20
2.3. XUL (Extensible User Interface Language).....	31
2.4. SVG (Scalable Vectorial Graphics)	41
2.5. UIML (User Interface Markup Language)	42
2.6. MXML (Macromedia eXtensible Markup Language).....	46
2.7. Herramientas libres y propietarias para la creación de interfaces de usuario multiplataforma. Herramientas para crear interfaces, editor XML.....	47
2.8. Edición del documento XML.....	48
3. Creación de componentes visuales:	50
3.1. Concepto de componente; características.	50
3.2. Propiedades y atributos.	50
3.3. Elementos Generales.....	51
3.4. Eventos; asociación de acciones a eventos.	54
4. Usabilidad.....	61
4.1. Concepto de usabilidad.	58
4.2. Medidas de usabilidad. Tipo de métrica.	61
4.3. Pautas de diseño de interfaces.	64
4.4. W3C	67
5. Confección de informes	70
5.1. Informes incrustados y no incrustados en la aplicación.....	70
5.2. Creación de parámetros	80
5.3. Creación de subinformes.....	81
5.4. Imágenes	82
5.5. Informes incrustados y no incrustados en la aplicación.....	82
5.6. Herramientas gráficas integradas y externas al IDE.	83
5.7. Filtrado de datos.	77
5.8. Numeración de líneas, recuentos y totales.	78
5.9. Librerías para la generación de informes. Clases, métodos y atributos.....	79

UF2. PREPARACIÓN Y DISTRIBUCIÓN DE APLICACIONES.....80

1. Realización de pruebas:	80
1.1. Objetivo, importancia y limitaciones del proceso de prueba. Estrategias.	87
1.2. Prueba unitaria.	81
1.3. Pruebas de integración: ascendentes y descendentes.	81
1.4. Pruebas del sistema: configuración, recuperación.	90
1.5. Pruebas de uso de recursos.	90
1.6. Pruebas de seguridad.	91
1.7. Pruebas funcionales.	91
1.8. Pruebas de simulaciones.	91
1.9. Pruebas de aceptación.	92
1.10. Pruebas alfa y beta.	92
1.11. Pruebas manuales y automáticas.	92
1.12. Herramientas de software para la realización de pruebas.	86
2. Documentación de aplicaciones:	87
2.1. Archivos de ayuda. Formatos.	94
2.2. Herramientas de generación de ayudas.	87
2.3. Tablas de contenidos, índices, sistemas de búsquedas, entre otros.	89
2.4. Tipos de manuales.	90
3. Distribución de aplicaciones.	92
3.1. Componentes de una aplicación. Empaquetado.	93
3.2. Instaladores.	94
3.3. Paquetes autoinstalables.	97
3.4. Herramientas para crear paquetes de instalación.	98

BIBLIOGRAFÍA 100

UF1. Diseño e implementación de interfaces

1. Confección de interfaces de usuario

Existen una serie de aplicaciones en los mercados que se utilizan para llevar a cabo la confección de interfaces. Dichas aplicaciones también reciben el nombre de **IDE** (Entorno de Desarrollo Integrado).

Entre sus principales características se encuentran las de **codificación**, **compilación**, **depuración** y **testeo** de los diferentes programas.

Las más importantes a destacar son: Visual Studio, NetBeans y Eclipse.

Visual Studio



Visual Studio es una IDE creada por Microsoft que se basa en centrar su núcleo de desarrollo próximo a esta solución.

Esta aplicación cuenta con diferentes versiones bajo licencia, aunque también ha desarrollado algunas versiones gratuitas con denominación *express*.

Estas últimas cuentan con menos funcionalidades y están más limitadas que las versiones con licencia, aunque disponen de un gran número de herramientas capaces de cubrir las distintas demandas del usuario particular.

- **Creación de un proyecto**

La creación de proyectos no es una tarea muy complicada en Visual Studio y, además, es un proceso bastante intuitivo. Desde que aparece la primera ventana ya es posible tener acceso a la creación de un proyecto y a todas sus opciones del menú Archivo.

Cuando se selecciona la opción *Nuevo Proyecto*, aparecerán una serie de plantillas de creación de proyectos clasificadas por su ámbito y por el lenguaje de programación que utilizan.

- **Área de trabajo**

Es posible diferenciar un área central de trabajo junto a otras ventanas flotantes que se pueden acoplar. Estas ventanas se pueden visualizar o no, según las necesidades del usuario. También es posible cambiarlas de lugar permitiendo al usuario personalizar su entorno de trabajo.

Además, el usuario tiene la posibilidad de diferenciar las áreas que van a formar parte de la interfaz, que suelen ser las mismas en las diferentes IDE.

- **Explorador de soluciones**

La ventana del explorador de soluciones aparece en la parte derecha del entorno gráfico del programa, aunque se puede modificar según las necesidades del usuario. Permite visualizar, de forma ordenada, los diferentes archivos que forman el proyecto.

- **Ventana de propiedades**

Diferencia entre la vista de propiedades y de eventos, es decir, puede mostrar un tipo de información u otra, dependiendo del elemento que seleccione el usuario en un determinado momento.

- **Menú**

En la opción **Menú**, es importante diferenciar entre las opciones en las que es posible guardar un trabajo que se haya desarrollado en el sistema de archivos → **Menú Archivo** y, la configuración determinada del proyecto en el que se esté trabajando → **Menú Proyecto**.

- **Menú archivo**

Permite generar nuevos proyectos, abrir los que ya existían, opciones para guardar, entre otras opciones.

- **Menú proyecto**

Permite gestionar las opciones disponibles de los distintos elementos que forman el proyecto.

NetBeans



Herramienta creada por **Sun Microsystems** que se basa en la utilización de código abierto, por lo que es libre y gratuita. Está desarrollada en el lenguaje de programación Java, aunque cuenta con la posibilidad de codificación de programas en otros lenguajes como XML, HTML, PHP, JavaScript, JavaDoc, Groovy y JSP.

La aplicación NetBeans puede instalarse en cualquier sistema operativo gracias a la capacidad de Java para poder verlo en diferentes programas.

Al instalar NetBeans, se instala una máquina virtual.

- **Creación de un proyecto**

Netbeans también cuenta con un listado de plantillas que permiten crear los diferentes proyectos. Estas se ordenan por categorías y, si se posiciona sobre cada una de ellas, es posible observar una pequeña información en la parte inferior.

- **Área de trabajo**

En esta parte es posible diferenciar **cuatro zonas principales**:

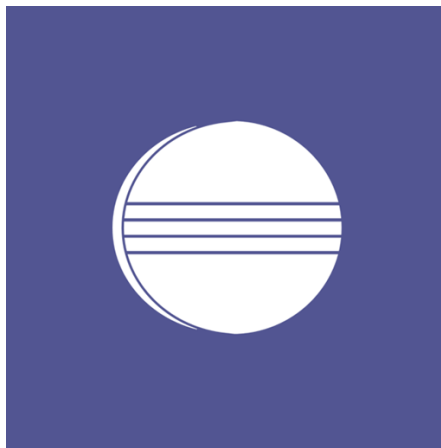
- En la primera se visualiza el **Menú principal** junto con la barra de herramientas.
- En la segunda, la **estructura** de los diferentes archivos que se han creado. Cuando existe un proyecto Java, los archivos se organizan en paquetes.
- La tercera zona corresponde al **área principal de trabajo** que es posible modificar según el tipo de proyecto sobre el que se esté trabajando.
- La última zona muestra los diferentes **métodos y propiedades** de la clase en cuestión.

- **Ventana de propiedades**

Ofrece al usuario la posibilidad de poder personalizar su ubicación en el entorno de trabajo que le ofrece NetBeans. Cada elemento cuenta con una gran parte de información que, al formar una interfaz, cuenta con una serie de vistas organizadas en distintas pestañas.

- **Propiedades.** Pestaña *properties*. Muestra las diferentes propiedades de un determinado objeto y ofrece la posibilidad de poder modificarlo haciendo uso de una serie de selectores, por ejemplo, los selectores de color.
- **Binding.** Esta pestaña ofrece la posibilidad de configurar las distintas fuentes de datos que necesitan las propiedades de un elemento.
- **Eventos.** Esta pestaña muestra aquellos eventos que se definen para el control. Además, ofrece la posibilidad de definir un tipo de manejador para una serie de eventos seleccionados.
- **Código.** En esta última pestaña es posible asignar un determinado especificador para un elemento determinado que se encuentre dentro del código.

Eclipse



Esta herramienta también está basada en código abierto y, aunque esté asociada al lenguaje Java, permite soportar otros lenguajes de programación y diferentes tipos de proyectos.

De la misma forma que *NetBeans*, esta herramienta está desarrollada en Java, aunque, en ocasiones, necesita hacer uso de alguna máquina que cuente con unas altas prestaciones para llevar a cabo, de forma correcta, su proceso de ejecución.

Eclipse ofrece la posibilidad de incluir nuevas funcionalidades adicionales a través de extensiones (plug-in).

La IDE de *Eclipse* diferencia una parte para el área de trabajo. En ella se pueden combinar los diferentes archivos organizados jerárquicamente (suele aparecer en la parte izquierda en una zona denominada editor), y se utiliza en el proceso de codificación o a la hora de diseñar interfaces que sigan el principio **WYSWYG**.

- **Creación de un proyecto**

Eclipse ofrece un asistente que guía al usuario para crear un nuevo proyecto. A lo largo de todas las ventanas que van apareciendo, se solicita información básica del proyecto que se desee crear.

Eclipse es una de las IDE más conocidas de entre los distintos grupos de desarrolladores y se utiliza a la hora de crear proyectos que estén basados en Java. Además, cuenta con una serie de plantillas extras que se pueden utilizar para crear proyectos en otros lenguajes de programación diferentes, por ejemplo, JavaScript, HTML5 o PHP, entre otros.

- **Área de trabajo**

La herramienta *Eclipse* ofrece la posibilidad de tener abiertos diferentes editores, organizados en ventanas, de tal manera que se le permite al usuario poder trabajar y poder cambiar de forma dinámica y nada complicada, la vista del área de trabajo en un momento determinado.

También es posible visualizar de manera alternativa tanto la interfaz seleccionada, como el código generado por dicha interfaz.

Cuenta con un entorno que es posible personalizar de forma sencilla, siendo el usuario el que puede decidir las ventanas de las que va a disponer, el orden de sus vistas y la posición que van a ocupar.

- **Explorador de paquetes**

Permite mostrar, de forma ordenada, las diferentes partes de un proyecto, ordenando los archivos necesarios de Java en paquetes.

En *Eclipse* se utiliza **Workspace**, el espacio en el que se van a organizar todos los datos de un proyecto determinado, además de una información del mismo, conocida como *metainformación*.

1.1. Librerías de componentes disponibles para los distintos sistemas operativos y lenguajes de programación. Características

Existe una gran variedad de librerías que permiten llevar a cabo el diseño de *interfaces*. Algunas pertenecen a un sistema operativo determinado, mientras que otras dependen del lenguaje de programación que se utilice.

- **AWT** (*Abstract Window Toolkit*):
 - Diseñada en Java puro y utilizada como base de la librería Swing.
 - Presenta un funcionamiento correcto a pesar de no contar con controles demasiado avanzados.
 - Utilizada para el desarrollo de diseños prácticos y eficientes.
 - Su aspecto dependerá del sistema operativo que utilice.
 - Con el paso del tiempo ha perdido protagonismo, por lo que hoy en día se considera obsoleta.

- **Swing**:
 - Diseñada en Java puro y creada a partir de la librería AWT.
 - Su función es dar respuesta a todos los inconvenientes que presente AWT.
 - Cuenta con controles de bastante funcionalidad utilizados para aplicaciones.
 - A pesar de mejorar muchas carencias, también cuenta con una serie de deficiencias, como pueden ser el filtrado y la organización de los datos (tanto en forma de árbol como de tabla). Por ello, es aconsejable desarrollar un manual de funcionalidad.

- **SWT** (*Standard Widget Toolkit*):
 - Creada por IBM para el entorno de desarrollo Eclipse.
 - Considerada la tercera generación de librerías (después de AWT y Swing) aunque no está muy relacionada con ellas.
 - Considerada una librería de bajo nivel que utiliza *widgets* nativos de la misma plataforma que ejecuta mediante JNI.
 - Estas interfaces no se pueden ejecutar en todas las plataformas.
 - Uno de sus inconvenientes es que la API que proporciona la librería es bastante complicada de utilizar y no muy intuitiva.

- **SwingX:**
 - Basada en Swing y utilizada para desarrollar aplicaciones RIA (*Rich Internet Application*).
 - Diseñada en Java puro.
 - Puede utilizarse en diferentes plataformas.

- **JavaFX:**
 - Desarrollada por Java/Oracle.
 - Plataforma Open Source.
 - Basada principalmente en el desarrollo de aplicaciones RIA.
 - Se puede usar en diferentes plataformas y dispositivos.

- **Apache Pivot:**
 - Librería de código abierto.
 - Principalmente utilizada en Apache Project.
 - Utilizada para desarrollar aplicaciones RIA (*Rich Internet Application*).
 - Basada en Java y otros lenguajes que actúan sobre la máquina virtual de Java.

- **Qt Jambi:**
 - Empaquetado para Java sobre la librería Qt.
 - Desarrollada en C++/C.
 - Al utilizar librerías nativas, no se puede usar en otras plataformas.
 - Tiene una gran aceptación y es bastante potente.
 - Es una interfaz bastante sencilla de utilizar.

- **Librerías OpenGL (*Open Graphics Library*):**
 - Encargada de definir una API multilenguaje y multiplataforma, con posibilidad de escribir aplicaciones que produzcan código 2D y 3D.
 - Dispone de un amplio número de funciones para crear gráficos en tres dimensiones.
 - Apta para creación de videojuegos, aeroespacial, etc.

- **API DirectX:**

- Grupo de APIs desarrolladas para simplificar las distintas tareas de vídeo y juegos.
- Utiliza la plataforma de Microsoft Windows.
- Utiliza una serie de versiones que mejoran las funciones de videojuegos.
- Aprovecha diferentes ventajas de arquitecturas de procesadores con varios núcleos, como:
 - **Direct2D:** desarrollada en C++ y permite generar los elementos gráficos.
 - **Direct3D:** API que genera el diseño y procesa las distintas interfaces en tres dimensiones.
 - **DirectCompute:** API que genera el diseño y procesa las distintas interfaces.

- **Qt:**

- Basada en las diferentes bibliotecas multiplataforma que permiten el desarrollo de interfaces gráficas.
- Creada y desarrollada en C++.
- Utiliza la programación orientada a objetos para hacer uso de otros lenguajes de programación.
- Permite el acceso a las bases de datos, utiliza XML, soporte de la red, etc.

- **GTK+:**

- Basada en las diferentes bibliotecas multiplataforma (Linux, Mac, Windows) que permiten el desarrollo de interfaces gráficas bajo un estándar de software libre (LGPL).
- Su desarrollo está basado en objetos.
- Muy utilizada por la interfaz GNOME.

1.2. Componentes: características y campos de aplicación

En los módulos anteriores de este Ciclo Formativo de Grado Superior y, especialmente, en el módulo 3 de Programación, se han desarrollado programas que usan la consola para interactuar con el usuario.

La interfaz en modo texto se caracteriza por ser muy simple y por el hecho de centrarse en todo aquello que tiene que ver con la programación orientada a objetos con el lenguaje Java, sin tener que tratar al mismo tiempo con ventanas, botones y otros elementos similares.

El siguiente paso para la realización de programas más intuitivos, atractivos y dinámicos es la utilización de interfaces gráficas de usuario (GUI) que ofrecen al usuario ventanas, cuadros de diálogo, barras de herramientas, botones, listas desplegables y otros elementos.

Además de los elementos, también se tratan las acciones asociadas a eventos que son posibles de realizar en dicha aplicación. Las aplicaciones se desarrollan haciendo uso de las clases que ofrece la API de Java.

Este lenguaje de programación y sus respectivas API o colecciones, proporcionan unas librerías que ya se han mencionado anteriormente, y que sirven para el desarrollo de interfaces gráficas de usuario (AWT y SWING). Las librerías proporcionan un conjunto de herramientas para facilitar la construcción de interfaces gráficas con una apariencia intuitiva y amigable.

La estructura básica de estas bibliotecas gira en torno a **componentes** y **contenedores**. Los contenedores a su vez lo forman componentes y son componentes a su vez, de forma que los eventos pueden tratarse tanto en contenedores como en componentes. La API está constituida por **clases**, **interfaces** y **derivaciones**.

1.3. Eventos, escuchadores y acciones a eventos

Además de los elementos, contenedores y métodos, los **eventos** son otra de las herramientas básicas para la implementación de las interfaces gráficas. La interactividad y la respuesta ante una entrada del usuario es posible gracias a los eventos y las acciones que se implementan cuando algunos de ellos ocurren.

Estos eventos son las interacciones del usuario con la interfaz. Cada componente está asociado a un evento. Será decisión del desarrollador el implementar una acción cuando ocurra el evento en ese componente.

Los objetos son las definiciones de la manera de tratar a los eventos. A estos objetos se les notifica la ejecución de un evento y automáticamente se relacionan con la acción definida. Se les denomina **Listeners**.

Existen dos tipos de eventos: **bajo y alto nivel**. Los primeros están relacionados con la interacción física del usuario, como, por ejemplo, **MouseEvent** o **WindowEvent**. Los segundos (alto nivel), por el contrario, se relacionan con las operaciones lógicas realizadas sobre los elementos, como, por ejemplo, la opción **Salir** (**ActionEvent**).

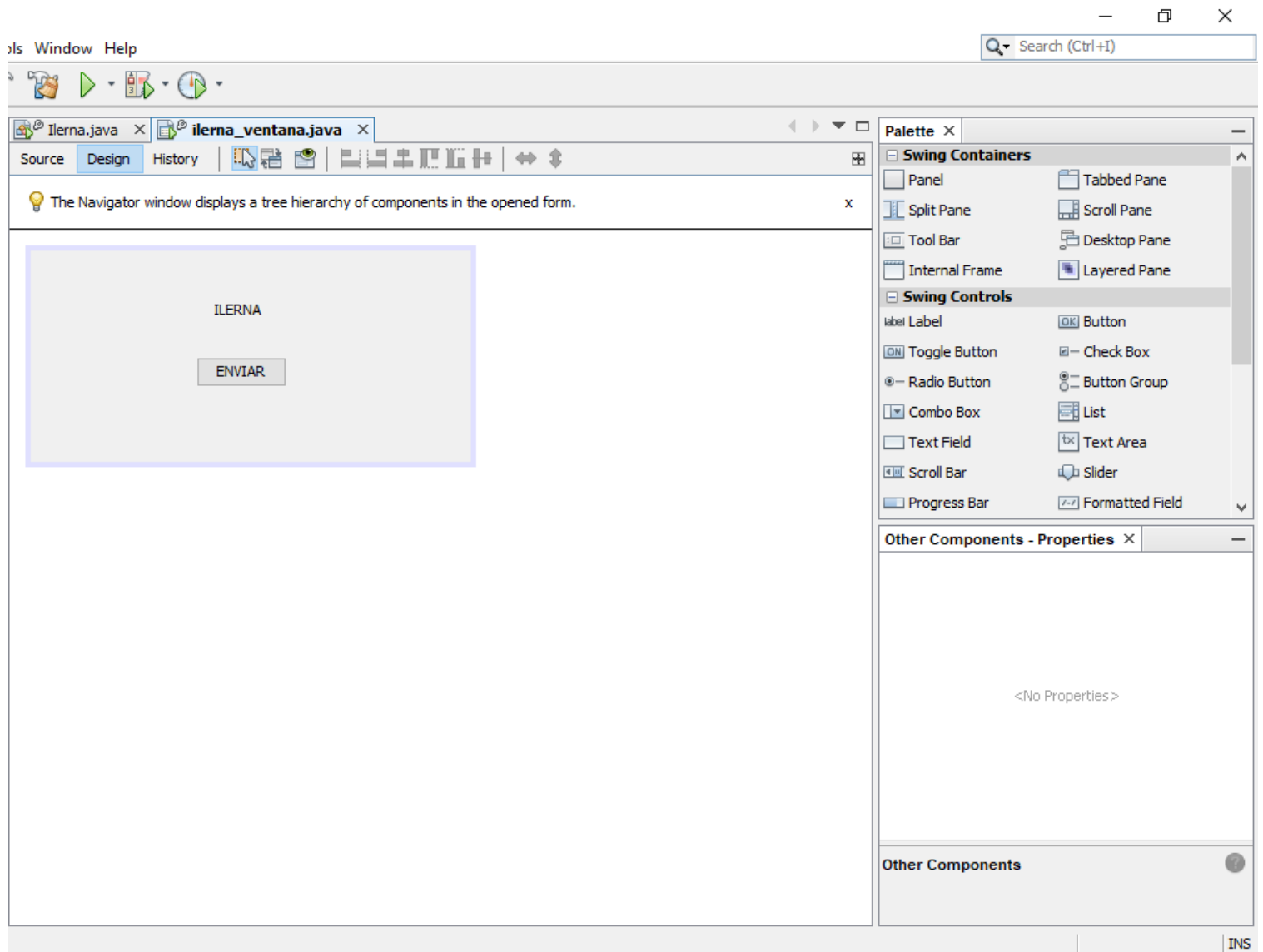
Para finalizar, el objeto listener tendrá que ser una clase que herede la interfaz correspondiente al tipo de evento que vaya a tratar.

1.4. Edición del código generado por las herramientas de diseño

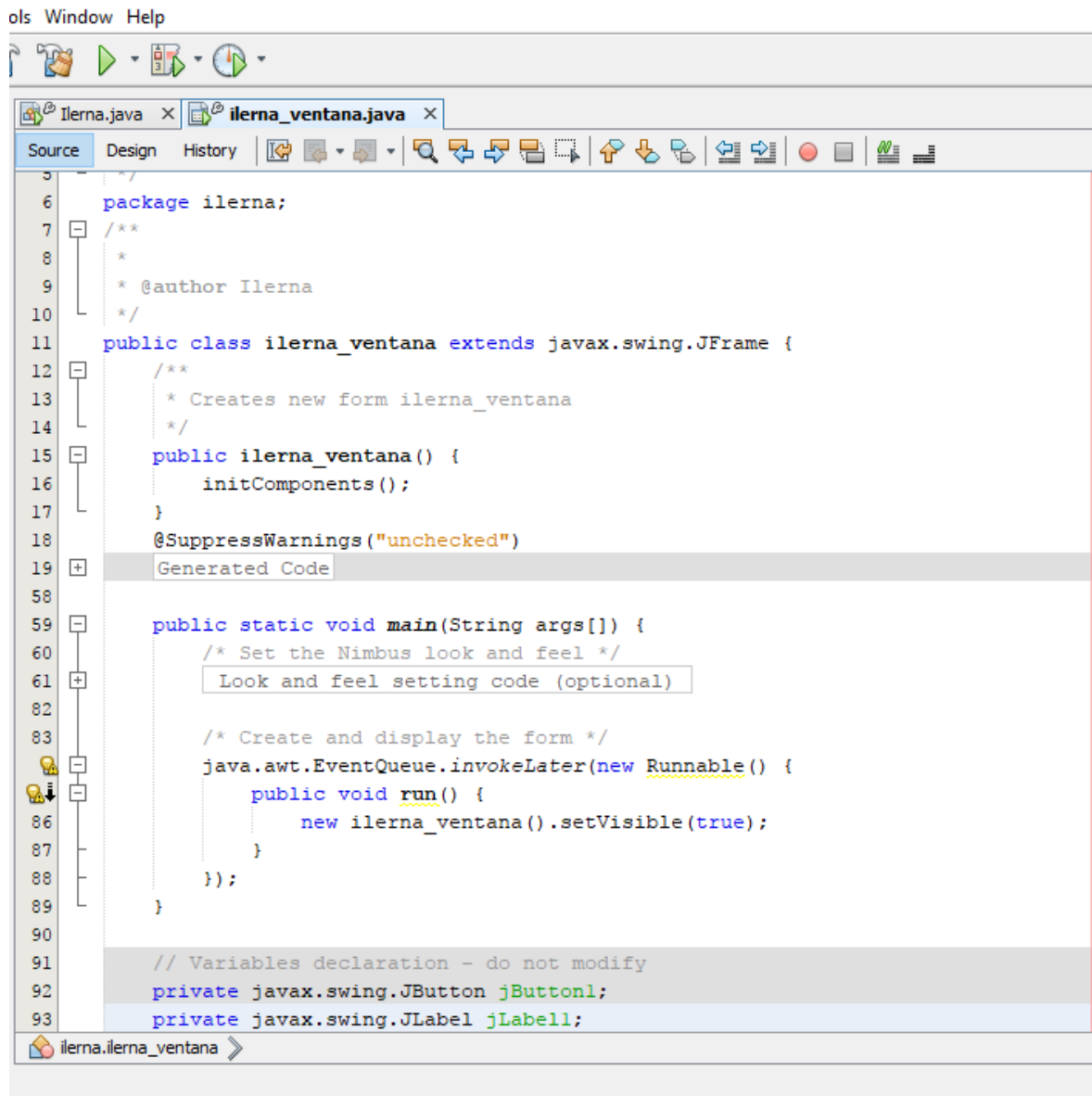
Para la creación de la interfaz gráfica es posible utilizar los *IDE* que anteriormente se han detallado.

Teniendo en cuenta el lenguaje *java* en cualquiera de las dos herramientas, los *IDE* facilitan la programación de una interfaz gráfica. Se dedica un apartado a la creación de la interfaz gráfica mediante una pestaña de diseño y en ella se pueden ir arrastrando los elementos deseados y automáticamente se traducen en el código correspondiente.

En la siguiente imagen es posible ver la vista **Diseño** a la que poder arrastrar los elementos deseados:



En la siguiente imagen se contempla el código correspondiente:

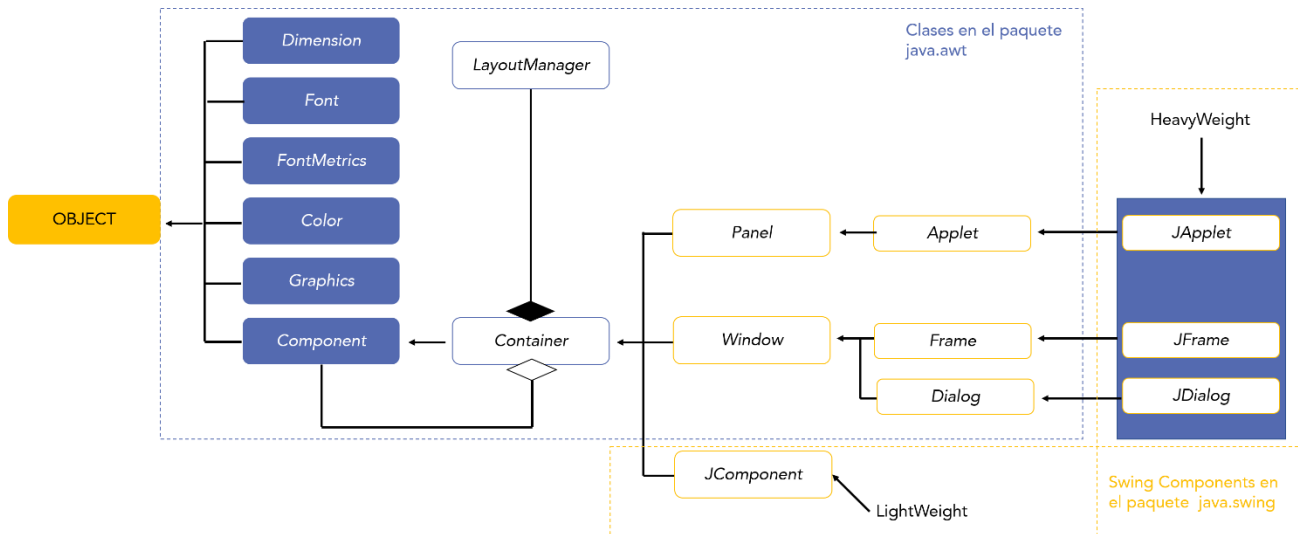


```

1  package ilerna;
2
3  /**
4   *
5   * @author Ilerna
6   */
7
8  public class ilerna_ventana extends javax.swing.JFrame {
9
10     /**
11      * Creates new form ilerna_ventana
12      */
13     public ilerna_ventana() {
14         initComponents();
15     }
16
17     @SuppressWarnings("unchecked")
18     Generated Code
19
20     public static void main(String args[]) {
21         /* Set the Nimbus look and feel */
22         Look and feel setting code (optional)
23
24         /* Create and display the form */
25         java.awt.EventQueue.invokeLater(new Runnable() {
26             public void run() {
27                 new ilerna_ventana().setVisible(true);
28             }
29         });
30     }
31
32     // Variables declaration - do not modify
33     private javax.swing.JButton jButton1;
34     private javax.swing.JLabel jLabel1;
35 }

```

1.5. Clases, propiedades, métodos

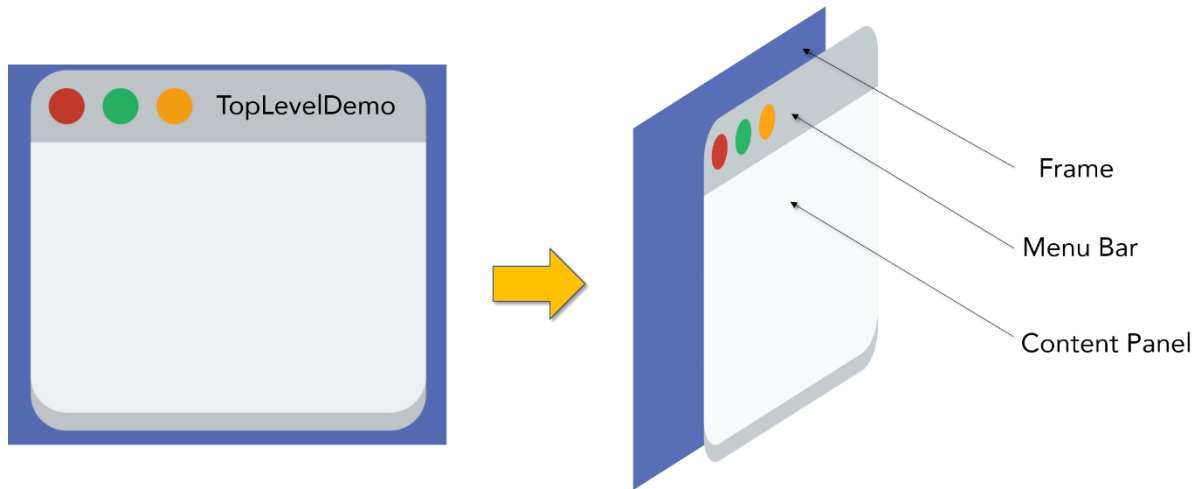


Como se ha explicado anteriormente, la creación de la interfaz gráfica se basa en la utilización de dos librerías. **Swing es más actual y se compone de clases y métodos más novedosos que AWT.**

Se utiliza un lenguaje de programación orientado a objetos, por ello, las clases más utilizadas en esta implementación con sus correspondientes funcionamientos son:

- **Component:** las librerías se forman a base de componentes. Por tanto, esta clase es la superclase de todas las clases de la interfaz gráfica.
- **Container:** todo componente se agrupa en contenedores, por tanto, esta es la clase contenedor.
- **JComponent:** es la clase *Component* pero en la librería *Swing*, ya que su nomenclatura identifica a sus clases con una "J" delante. En ella se dibujan directamente los lienzos (*canvas*). Sus subclases son los elementos básicos de la **GUI**, que son:
 - **JFrame:** ventana que no está contenida en otras ventanas. Normalmente se utiliza para crear la ventana principal de la aplicación.
 - **JDialog:** cuadro de diálogo.
 - **JApplet:** subclase de *Applet* para crear applets tipo *Swing*.
 - **JPanel:** contenedor invisible que mantiene componentes de interfaz y que se puede anidar, colocándose en otros paneles o en ventanas. También sirve de lienzo.
 - **JScrollPane:** panel con barras de desplazamientos.

- **Graphics:** clase abstracta que proporciona contextos gráficos donde dibujar cadenas de texto, líneas y otras formas sencillas.



Las librerías se componen de clases y cada clase tiene unos métodos para su utilización a la hora de crear interfaces gráficas.

A continuación, mediante el enlace se pueden comprobar todos los métodos de cada clase en la dirección de la API correspondiente:

<https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

Además, es posible crear clases propias para el diseño de la interfaz gráfica, donde hay que heredar una clase que sí esté contenida en la librería gráfica en cuestión.

Por ejemplo:

```
class Miventana extends JFrame
{
    JPanel panel; // atributos
    JLabel label;

    // constructor
    Miventana( String title )
    {
        super( title );// invoca al constructor de JFrame
        setSize( 150, 100 );
        setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        setLayout( new FlowLayout() );
        label = new JLabel("Hola ventana!");
        add( label );

    }

}
```

2. Generación de interfaces a partir de documentos XML

2.1. Lenguajes de descripción de interfaces basadas en XML. Ámbito de aplicación

Para llevar a cabo el diseño de interfaces, es necesario tener en cuenta que, en este tipo de diseños, van a predominar elementos como: botones, cajas de texto, o listas desplegables, entre otros. Todos ellos son diferentes formas que existen para poder representar los distintos datos.



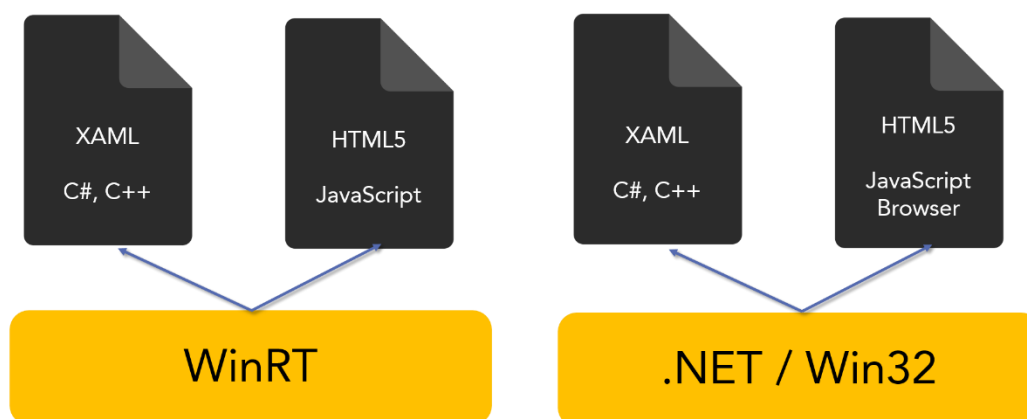
Es importante conocer que, la interfaz de usuario, aparte de los elementos comentados anteriormente, también cuenta con un conjunto bastante amplio de elementos gráficos como pueden ser, entre otros, los diferentes colores, degradados y figuras en 2D y 3D.

Aquellos elementos que sean completamente gráficos se pueden diseñar y representar en los mismos lenguajes que se utilizan para programación, definiendo los correspondientes elementos de entrada/salida, aunque también es posible dar uso a las librerías determinadas.

2.2. XAML (*Extensible Application Markup Language*)

Este lenguaje está basado en XML y permite realizar una descripción gráfica de las interfaces de los distintos usuarios (desde el punto de vista gráfico).

XAML es un lenguaje que se puede aplicar al lenguaje de desarrollo de interfaces para escritorio y, además, suele utilizarse para web. Existen una serie de editores que permiten incorporar herramientas de edición y analizadores sintácticos, como pueden ser *Visual Studio* y *Blend* entre otros.



Uno de los principales objetivos que se pretende en el diseño de interfaces que están basadas en XAML es **separar totalmente las capas de presentación de la capa lógica**, para conseguir evitar que se mezclen aquellos elementos que pertenezcan a distintas capas. Esto podría afectar a la distribución modular de la aplicación y al acoplamiento de la misma.

A la hora de tener que ilustrar la estructura de una interfaz basada en XAML, se diseña una pequeña interfaz que dispone de un botón en la misma. Es por esto que se necesitaría crear un nuevo proyecto en el editor *Visual Studio* haciendo uso del lenguaje de programación C#, por ejemplo.

A continuación, aparecerá una ventana inicial del proyecto en la que se puede apreciar en una ventana el trozo de código XAML y, en otra, la interfaz resultante.

Se debe comprobar que el elemento más importante es la etiqueta **<Window>** que va a ser el elemento raíz y debe finalizar con **</Window>**. Detrás de esta etiqueta ya no se puede generar código adicional.

Dentro de la etiqueta *Window* es preciso declarar los distintos espacios de nombres junto con sus correspondientes referencias.

Estos espacios de nombres deben asociar los elementos descritos en el documento con los determinados controles de **WPF** (*Window Presentation Foundation*) que están en el espacio de nombres *System. Windows. Controls* del *.Net Framework*.

Entre las principales características de XAML se puede señalar que **cada elemento gráfico se define mediante una etiqueta de apertura y otra de cierre, además de un conjunto de atributos, que definirán el aspecto y comportamiento del mismo.**

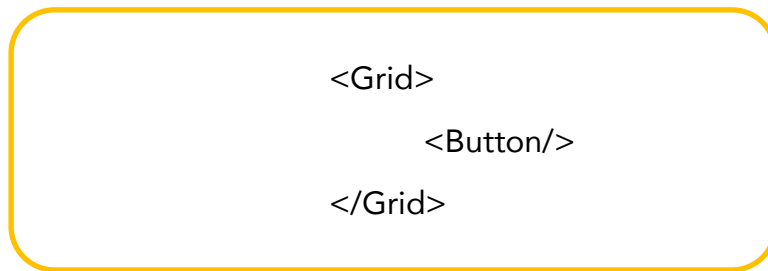
XALM cuenta con bastantes **ventajas** respecto a sus competidores, sobre todo, al permitir desarrollar distintas interfaces mediante su asociación con *.Net*. En este lenguaje, tanto las etiquetas como los atributos se corresponden de forma directa con otros elementos que pertenecen al lenguaje *.Net*.

Descripción sintaxis en XAML

Las etiquetas XAML definen los distintos elementos pertenecientes a la interfaz y cuentan con una serie de atributos:

- **Atributo *Name***: único identificador del elemento. Bastante útil cuando se precisa hacer referencias en el código (***x>Name***).
- **Atributo *Key***: puede contar con un identificador para los distintos elementos definidos en el diccionario (***x:Key***).
- **{*Binding*}**: se refiere al elemento que está definido dentro de un valor de un atributo, permitiendo definir un enlace a una fuente de datos, un fichero, una base de datos, etc.
- **{*StaticResource*}**: similar al anterior, pero, en este caso, referencia a un elemento que está definido en el diccionario de recursos.
- **{*Null*}**: representa el valor nulo.

Como ejemplo, la forma de representar un botón:



Existen dos formas para realizar una representación con sintaxis, partiendo de la estructura anterior:

1. **Forma 1.** Basada en atributo.

Sencilla pero no muy manejable ya que carece de fuerza.

```
<Button Width="125" Height="75">
```

Aceptar

```
</Button>
```

2. **Forma 2.** Basada en propiedad.

Utiliza elementos gráficos, por lo que su representación es más compleja.

```
<Button Width="125" Height="75">
```

```
<Button.Background>
```

...

```
</ButtonBackground >
```

Aceptar

```
</Button>
```

Contenido de texto

Se muestra a continuación la manera de determinar el texto de un botón a través de las diferentes configuraciones que están basadas en atributo, junto con dos formas de representar lo mismo:

```
<Button Width="100" Height="75">
```

```
    <Button.Content>Aceptar</Button.Content >
```

```
</Button>
```

```
<Button Width="100" Height="75"Content="Aceptar"></Button>
```

A continuación, se usa el atributo *content* en lugar de añadir el texto deseado entre las etiquetas del elemento *Button*. Es conveniente indicar que el contenido de texto se debe colocar unido, no por partes:

```
<Button>Contenido de
```

```
    <Button.Background>
```

```
        <SolidColorBrush Color="Red" />
```

```
    </Button.<background>texto
```

```
</Button>
```

El propio compilador debe detectar un error al comprobar que existen varias definiciones del contenido cuando solo debe haber una.

Las diferentes posibilidades que ofrece **XAML** se indican a continuación:

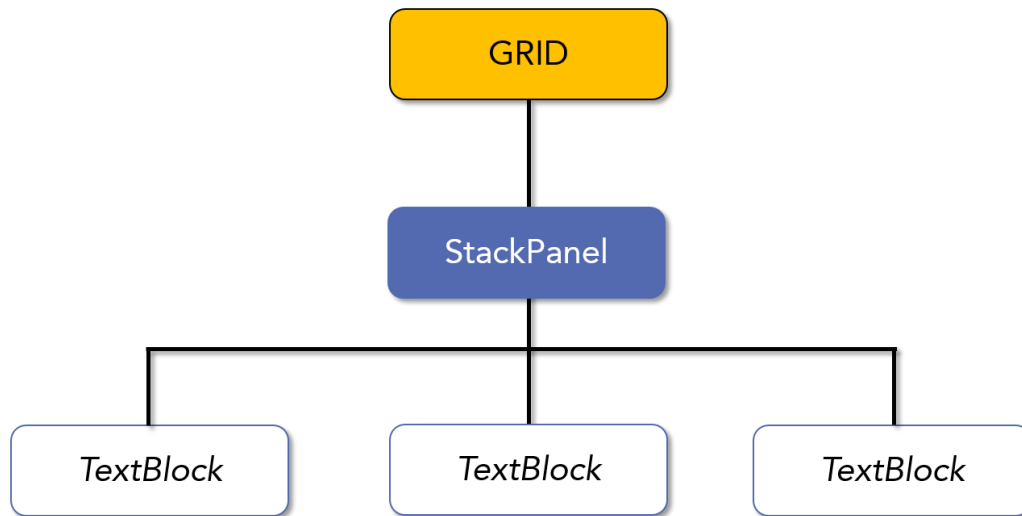
```
<Button Width="100" Height="75">
    <Button.Background>
        <SolidColorBrush Color="Red" />
    </Button.Background>
    Contenido texto
</Button>
```

```
<Button Width="100" Height="75">
    Contenido texto
    <Button.Background>
        <SolidColorBrush Color="Red" />
    </Button.Background>
</Button>
```

Representación de contenedores

Cuando se está desarrollando una interfaz y se busca su validez para todo tipo de entornos, es conveniente hacer uso de una serie de estructuras denominadas **contenedores**. Gracias a ellos, es posible tener organizada toda la información.

Existen cinco tipos diferentes de contenedores dentro de XAML que se muestran en la siguiente imagen:

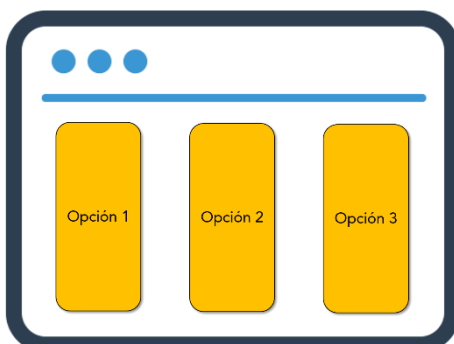


A continuación, se detalla cada uno de ellos, especificando su funcionamiento y la salida que genera:

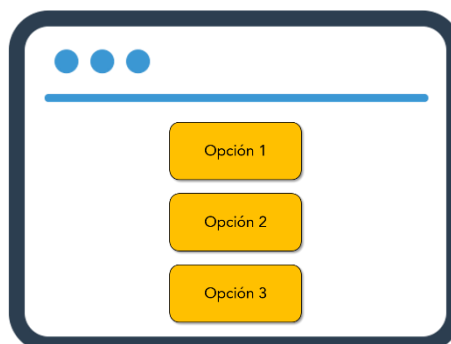
- **StackPanel**

Muy utilizado a la hora de diseñar listas, ya que ofrece la posibilidad de apilar los diferentes elementos de dos formas diferentes:

- **Orientación horizontal:** uno al lado de otro.
- **Orientación vertical:** uno encima de otro.



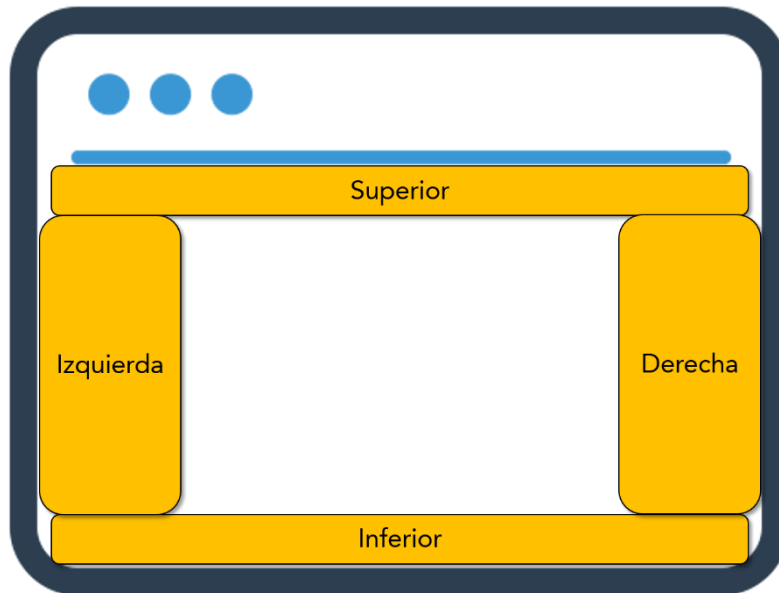
Orientación horizontal



Orientación vertical

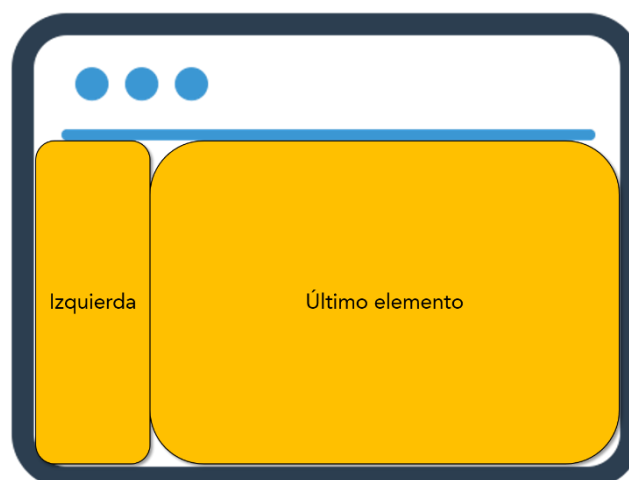
- **DockPanel**

Ofrece la posibilidad de anclaje de los diferentes elementos en los márgenes izquierdo, derecho, superior e inferior, situándose en un lugar determinado.



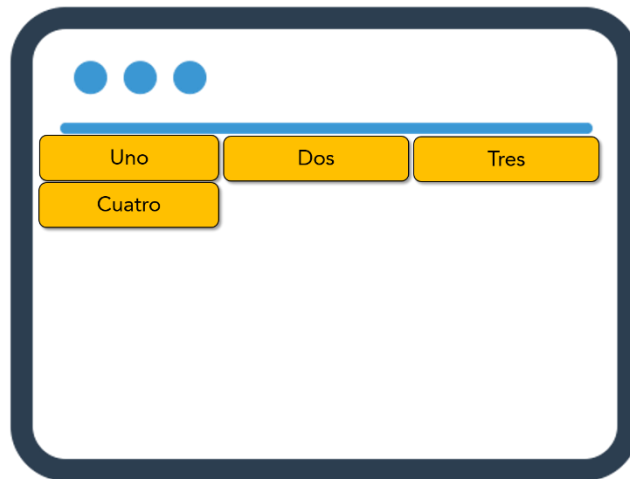
Existe una opción muy utilizada que permite determinar un elemento que ocupe todo el contenido, sobre todo para el explorador de archivos y, en la parte izquierda, las opciones del menú.

Para ello, es necesario utilizar la propiedad **LastChildFill**.



- **WrapPanel**

Permite unificar los distintos elementos orientándolos de manera vertical u horizontal, aunque esto solo es posible para aquellos elementos que caben en una fila.



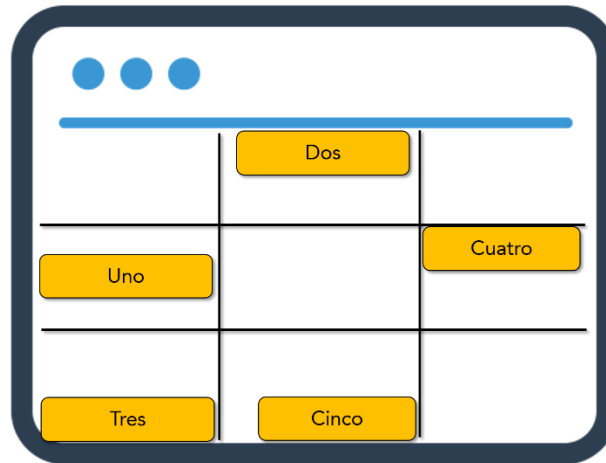
- **Canvas**

Es el elemento más utilizado y también el que cuenta con un mayor número de opciones. Ofrece la posibilidad de agrupar los elementos en determinadas coordenadas, aunque estas también se pueden determinar mediante las coordenadas relativas, haciendo uso de las referencias *Canvas.Left*, *Canvas.Right*, entre otras.



- **Grid**

Permite declarar los elementos ordenándolos en forma tabular: filas y columnas. Tiene un comportamiento bastante parecido al objeto tabla de HTML pudiendo unir celdas, columnas, etc. Cuenta con la posibilidad de determinar la alineación vertical y horizontal de cada celda, además de otros elementos.



Como resumen

Todos los contenedores que proporciona XAML deben seguir las especificaciones siguientes:

1. **No utilizar posiciones fijas** a través de las coordenadas absolutas. Es más conveniente utilizar los atributos *Alignment* y *Margin* para asegurar su correcto funcionamiento.
2. **No asignar tamaños fijos** a los elementos, ya que es preferible hacer uso de la propiedad auto.
3. Utilizar el elemento **Canvas solo cuando sea necesario**.
4. Utilizar el contenedor ***stackPanel*** en los botones de diálogo.
5. Utilizar el contenedor ***GridPanel*** a la hora de definir una interfaz de entrada de datos estática.

Cuadros de diálogo

Los cuadros de diálogo definen la relación que existe entre la interfaz y el usuario. Son ventanas de pequeño tamaño, a modo de ventanas emergentes, que permanecerán abiertas hasta que el usuario decida cerrarlas. Mientras la ventana está abierta, el usuario no puede interactuar con la ventana principal.

Las principales son:

- 1. Mensajes**
- 2. Controles de Diálogo**

Los botones de diálogo suelen estar asociados a estas ventanas para notificar, mediante la pulsación, una respuesta de la ventana de diálogo.

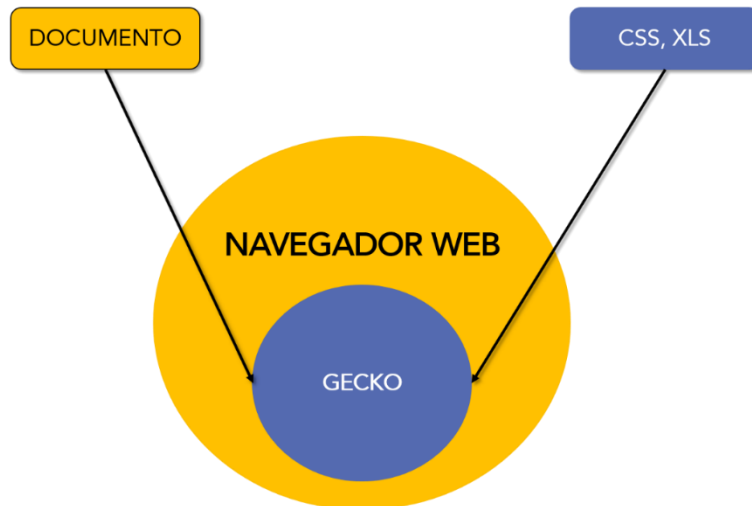
Flujo de eventos

A través del flujo de eventos es posible especificar el funcionamiento y la propagación de los diferentes eventos, es decir, se puede realizar la captura de los elementos lanzados mediante *eventos burbuja*.

Un evento burbuja permite describir cómo un contenedor se puede capturar y manejar por el propio contenedor.

2.3. XUL (*Extensible User Interface Language*)

Aunque hoy en día sigue desarrollándose por parte de **Mozilla**, es posible comprobar el funcionamiento de este lenguaje comunmente denominado **Gecko**.



Toma datos tipo HTML, XML y diferente información de formato tipo CSS, XLS para, a continuación, visualizar en pantalla el resultado que se ha obtenido tras aplicar el formato a los distintos datos.

Contenedores

Este lenguaje cuenta con una serie de contenedores definidos en *XUL* por el control *box*, que, a su vez, puede contener otros elementos. Casos concretos de control *box*:

- **Boxes**

En lo referente al modelo de cajas, es uno de los contenedores más utilizados definidos por XUL. Está basado en un principio que permite llevar a cabo la división de ventanas en diferentes cajas, posicionando los distintos elementos dentro de una de ellas. Estos elementos se denominan **hijos** y pueden situarse de manera vertical u horizontal.

La interfaz que se desarrolla puede obtenerse a través de:

- Una agrupación de estructuras de cajas.
- La orientación de los hijos dentro de las cajas.
- La aplicación de las distintas hojas de estilo.
- Existe otra opción en la que se aplican distintos elementos `<spacer>` con atributos *flex* y *pack*.

- **Sintaxis**

La sintaxis correspondiente a la definición de una marca de elementos con orientación **horizontal** mediante XUL se puede llevar a cabo de la siguiente forma:

```
<hbox>
    <!--horizontal -->
</hbox>
```

Los elementos se pueden posicionar de forma similar a una fila, mediante la utilización de un objeto **<table>** de HTML, añadiendo cada elemento por la derecha, aumentando, de esta manera, el ancho de la caja correspondiente.

Si lo que se pretende es llevar a cabo la ordenación mediante una **orientación** vertical, la sintaxis debe ser:

```
<vbox>
    <!--vertical -->
</vbox>
```

En este caso, los distintos elementos se colocan por columnas, añadiendo los nuevos en la parte inferior. De esta forma, irá aumentando la anchura de la caja.

En XUL también existe la posibilidad de añadir elementos genéricos de caja, mediante la utilización de los diferentes atributos. De esta forma se consigue el mismo funcionamiento que en las etiquetas anteriores **<vbox>** y **<hbox>**, por lo que se utilizará el atributo *orient* con los valores “horizontal” o “vertical”, según interese para cada situación.

En el siguiente ejemplo se muestran dos elementos iguales:

```
<vbox></vbox>
<box orient="vertical"></box>
```

- **Stacks**

Es bastante parecido al elemento *box*, ya que los elementos hijos se colocan unos encima de otros.

El elemento hijo que tenga un mayor tamaño será el que determine el tamaño del elemento. No obstante, también es posible definirlo mediante las hojas de estilo, especificando sus valores tanto de alto como de ancho.

- **Decks**

Similar al elemento anterior salvo que, en este caso, **Decks solo puede visualizar el primer elemento hijo.**

A continuación, se muestra un ejemplo en el que se definen tres elementos (páginas). La página seleccionada por defecto será, en este caso, la tercera, ya que está basada en **zero- index**.

Para cambiar de una página a otra, será necesario utilizar *javascript* cambiando el atributo **selectedIndex**:

```
<deck selectedIndex="2">
  <description value="Esta es la primera
  página"/>
  <button label="Esta es las segunda
  página"/>
  <box>
    <description value="Esta es la tercera
    página"/>
    <button label="Esta también es la
    tercera página"/>
  </box>
</deck>
```


También existe un elemento bastante importante a tener en cuenta como es el hecho de poder definir las distintas coordenadas relativas, que referencian a aquellos elementos hijos dentro de un contenedor. Para ello, es preciso hacer uso de los atributos **left** y **top**.

```
<stack>
  <button label="Portero" left="5"
top="5"/>
  <button label="Delantero" left=""60
top="20"/>
  <button label="Defensa" left="10"
top="60"/>
</stack>    <description value="Esta es
la tercera página"/>
  <button label="Esta también es la
tercera página"/>
  </box>
</deck>
```

Cuando se trata de una pila, el tamaño ya viene determinado por las distintas posiciones de sus elementos hijo, que son los que siempre van a ser visibles. Estos valores se pueden alterar utilizando las hojas de estilo.

- **Grid**

Mediante *XUL* es posible originar una salida similar a una tabla, almacenando las propiedades principales con el elemento en cuestión *<table>* de HTML, donde los distintos elementos se pueden situar en las diferentes filas y columnas. Por lo tanto, se cuenta con una serie de elementos fundamentales, como son los **atributos filas** (*rows*) y **columnas** (*columns*), donde se sitúan los elementos que disponen de la información que se desea mostrar.

A continuación, se lleva a la práctica un breve ejemplo para ver cómo resolverlo.

Se pretende crear una salida en la que aparezcan los botones de:

conejo, elefante,

koala y gorila

En el mismo orden en el que se ve en el enunciado:

```
<grid flex="1">
  <columns>
    <column flex="2"/>
    <column flex="1"/>
  </columns>
  <rows>
    <row>
      <button label="Conejo"/>
      <button label="Elefante"/>
    </row>
    <row>
      <button label="Koala"/>
      <button label="Gorila"/>
    </row>
  </rows>
</grid></stack>    <description
value="Esta es la tercera página"/>
  <button label="Esta también es la
tercera página"/>
</box>
</deck>
```

- **Extensión de columnas**

De la misma forma en la que se opera con el elemento `<table>` de HTML, es posible realizar la misma función mediante XUL.

Dispone de una forma de programar bastante sencilla en la que se coloca el elemento deseado fuera de las correspondientes etiquetas, aunque dentro de la etiqueta `<rows>`. El botón va a llevar a cabo todo el ancho del elemento contenedor **Grid**. Esta situación sería la misma si en lugar de hacerlo en las filas correspondientes, se hiciera para las columnas.

```
<grid>
  <columns>
    <column flex="1"/>
    <column flex="1"/>
  </columns>
  <rows>
    <row>
      <label value="Noroeste"/>
      <label value="Noreste"/>
    </row>
    <button label="Ecuador"/>
    <row>
      <label value="Sureste"/>
      <label value="Suroeste"/>
    </row>
  </rows>
</grid></stack>    <description value="Esta
es la tercera página"/>

  <button label="Esta también es la tercera
página"/>
</box>
</deck>
```

- **Tabboxes**

Estos elementos se suelen utilizar para organizar la distinta información a través de contenedores diferentes, indicando cada uno de ellos mediante diversas pestañas. De esta forma, el usuario podrá seleccionar la pestaña que desee y, una vez en la ventana elegida, podrá tener acceso a la información.

Existen una serie de elementos que forman parte de Tabboxes:

- **Tabbox**: es el elemento más externo y contiene al resto de elementos.
- **Tabs**: es el elemento contenedor de las distintas pestañas.
- **Tab**: se refiere a la pestaña que va a determinar que forma parte de una colección anterior.
- **Tabpanels**: es el elemento contenedor de las páginas.
- **Tabpanel**: página determinada de la colección tabpanels. Se refiere al cuerpo de una página.

A continuación, se detalla un ejemplo en el que es posible ver la forma de implementar este elemento:

```
<tabbox id="tablist">
  <tabs>
    <!--contenido-->
  </tabs>
  <tabpanels>
    <--contenido -->
  </tabpanels>
</tabbox></stack>    <description
value="Esta es la tercera página"/>
  <button label="Esta también es la tercera
página"/>
</box>
</deck>
```

El tamaño de *tabbox* está definido por su elemento interno de mayor tamaño.

- **Selección**

El atributo *selected*, cuando tiene valor cierto (*true*), puede seleccionar por defecto una pestaña, o puede llevar a cabo la selección a través de *javascript*.

Solo va a haber un elemento que tenga el valor *true*.

- **Posición**

Es posible determinar la posición de las distintas pestañas de un elemento.

Para tal fin, se cuenta con los atributos *orient*, *vertical* y *horizontal*, además del atributo *dir* con su correspondiente *valor reverse*.

- **Paneles**

A través de estos elementos es posible añadir en la interfaz información que procede de otras páginas o de otros documentos.

Es posible utilizar un elemento determinado para añadir elementos procedentes de páginas diferentes, como puede ser **iframe**. Su uso es bastante parecido al de HTML, con la salvedad de que **iframe** permite incluir otros archivos en cualquier parte de la interfaz, siempre que se indique el lugar correspondiente.

- **Browsers**

Es posible crear una integración con *browser*, por lo que el elemento en cuestión funcionará de forma muy parecida a un navegador.

El elemento **tabbrowser** está asociado a *browser* desarrollando un comportamiento muy parecido al *tabbox*, permitiendo modificar las distintas páginas en las que existe un browser.

- **Cuadros de diálogo**

Llegado el momento de definir los cuadros de diálogo, se deja de utilizar el elemento raíz de Windows y se pasa a utilizar la etiqueta **<dialog></dialog>** que ocupará la posición del elemento raíz.

En las ventanas de diálogo es posible representar diferentes botones, siendo los más utilizados los de *Aceptar* y *Cancelar*.

El número de botones que se pueden mostrar en las ventanas de diálogo están definidos mediante el atributo **buttons**, como, por ejemplo:

```
<dialog
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"

Id="dcg" tittle="MiDialogo"

buttons="aceptar, cancelar"

ondialogaccept="return onAccept();"

ondialogaccept="return
onCancel();"></stack>    <description
value="Esta es la tercera página"/>

    <button label="Esta también es la
tercera página"/>
```

Existen diferentes tipos de botones para los cuadros de diálogo dentro de XUL.

Tipo	Función
accept	OK
cancel	Cancelar
Disclosure	Más información
Help	Ayuda
Extra1, Extra2	Sin función preestablecida

- **File Picker**

Mediante la utilización de este elemento es posible elegir los archivos o directorios correspondientes. Presenta un comportamiento como ventana modal, en la que las funciones que se presentan dependerán de un método determinado.

- **Open:** el usuario desea abrir un archivo.
- **GetFolder:** el usuario desea elegir una determinada carpeta.
- **Save:** el usuario desea guardar un archivo.

A continuación, se expone un fragmento de código que detalla la forma de crear un objeto tipo *FilePicker*:

```
var
miFilePicker=Components.interfaces.nsIFileP
icker;

var
fp=Components.classes["@mozilla.org/filepic
ker;1"].createInstance(miFilePicker);

fp.init(window,"Seleccionar un
archivo",miFilePicker.modeOpen);  </box>

</deck>
```

La ventana modal asociada a *filepicker* cuenta con la opción de devolver tres valores diferentes que informan sobre la función desempeñada.

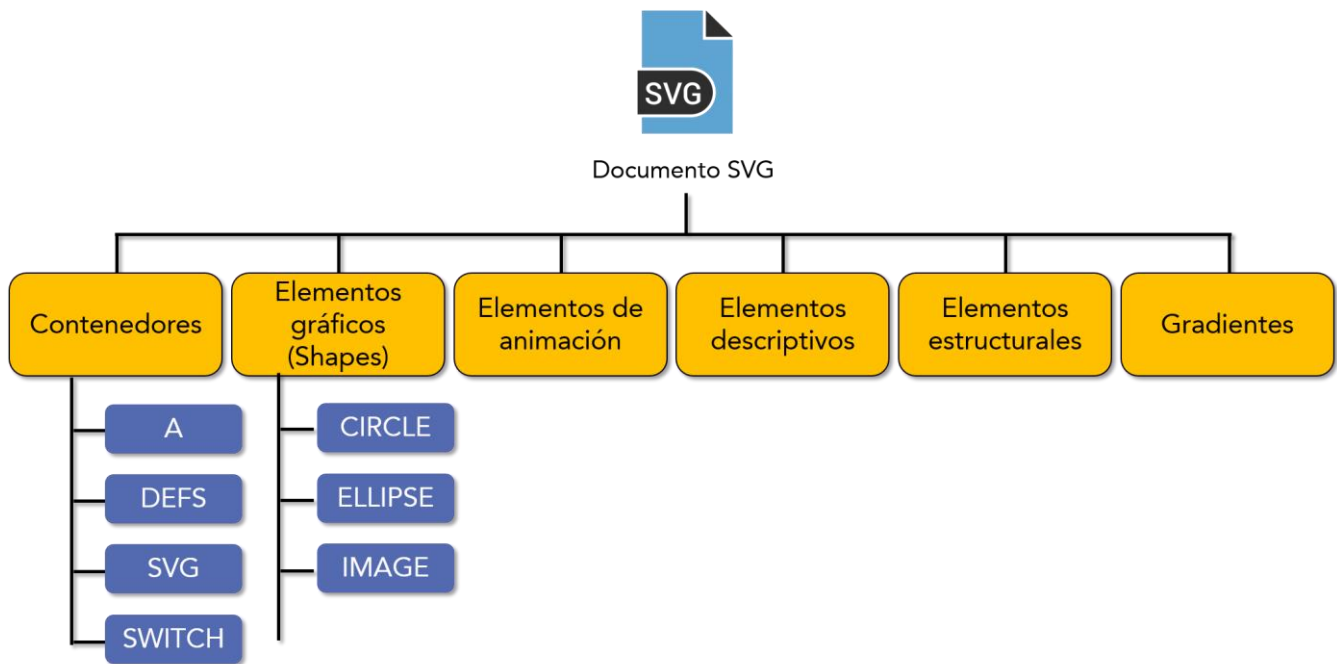
1. **returnOK:** el usuario elige un archivo y pulsa ok. El nombre del archivo referenciado se va a guardar en la propiedad *file* del elemento *file picker*.
2. **returnCancel:** el usuario pulsa el botón de cancelar.
3. **returnReplace:** para reemplazar un archivo es posible seleccionar el que va a sustituirlo y sobreescribirlo.

2.4. SVG (Scalable Vectorial Graphics)

Lenguaje basado en XML que se emplea para **definir gráficos vectoriales en 2D**.

SVG ofrece la posibilidad de definir distintas figuras simples o complejas que son posibles de especificar a través de ecuaciones matemáticas o incluso expresiones algebraicas.

En la imagen se muestra la **estructura** que deben tener los documentos SVG:



En la imagen se pueden distinguir:

1. **Elementos contenedores:** contienen, a su vez, otros elementos SVG. Esto facilitará la ramificación y escalabilidad de los documentos SVG.
2. **Elementos gráficos:** están formados por varios segmentos rectos y curvas, como pueden ser los círculos, rectángulos, etc. Permiten realizar representaciones en 2D o imágenes para “pegar” diferentes fotos o gráficos.
3. **Elementos de animación:** distintos efectos que se pueden aplicar a otros elementos SVG, de tal manera que garantizan dinamismo al conjunto.
4. **Elementos descriptivos:** ofrecen más información referente al elemento que los contiene (padre).
5. **Elementos de gradiente:** elementos SVG que se emplean para la definición de los gradientes de color que se pueden aplicar a otros elementos.
6. **Elementos estructurales:** elementos que definen la estructura primaria de los documentos SVG: *'defs'*, *'g'*, *'svg'*, *'symbol'*, *'use'*.

- **Referencias**

El lenguaje SVG permite llevar a cabo una serie de referencias a distintos elementos, de tal forma que se pueden aplicar sobre ellos los distintos efectos y animaciones.

También permite realizar esas animaciones o transformaciones sobre otros elementos que estén dispersos en el documento SVG mediante sus correspondientes referencias.

- **Agrupación de elementos**

SVG permite la agrupación de varios elementos en un mismo contenedor, de tal manera que es posible realizar distintas acciones, aplicar animaciones, entre otras, sobre todos los elementos.

- **Atributos de evento (*event attributes*)**

Pueden especificar el *script* que se debe ejecutar para un determinado evento que esté asociado a un objeto SVG.

Gracias a ellos, es posible especificar, por ejemplo, qué hay que ejecutar cuando se aplica una animación a un objeto (*onbegin*), cuándo va a finalizar la animación (*onend*), etc.

Existen diferentes *scripts* que se encargan de cambiar el contenido de un documento SVG o le añaden animaciones o transformaciones sobre los elementos que contiene. Existen dos maneras de reliarlo:

1. Mediante **funciones de DOM** o si el documento SVG es considerado XML. En este caso, se hace uso de DOM para manipularlo.
2. Si, por otro lado, se considera el documento como un SVG es posible utilizar el **modelo de objetos propios de SVG** para poder manipularlo.

2.5. UIML (User Interface Markup Language)

Fue creado por software *Harmonia*, compañía que ha publicado **UIML** como un lenguaje de código abierto.

UIML es un lenguaje descriptivo que ofrece la posibilidad de crear una página web para utilizarla en cualquier tipo de interfaz o dispositivo.

Entre sus **ventajas** principales es posible destacar las siguientes:

1. El **contenido web se debe crear solo una vez** y no necesita conocer las características del dispositivo que lo vaya a utilizar o visualizar.
2. Es conveniente que el desarrollador haga uso de un **lenguaje de marcado** a la hora de realizar la descripción de los elementos de la interfaz.

UIML se basa en un lenguaje de marcado extensible (XML). Necesita determinar una serie de elementos de la interfaz de usuario (*widgets*), por lo que es conveniente identificar el conjunto de elementos existentes y sus propiedades.

- **Sintaxis**

Tiene una estructura muy parecida a la de un documento *HTML*, salvo que, en este caso, se emplea la etiqueta `<UIML></UIML>` para apertura y cierre.

Existen tres tipos de secciones en los documentos UIML:

1. **HEAD**: es la parte que contiene la información referente al propio archivo, como pueden ser el autor, la fecha y la versión.
2. **APP**: esta sección contiene la estructura correspondiente a la interfaz y cuenta con dos elementos para diseñarla, que son `<GROUP>` y `<ELEM>`.

`<APP>` necesita el atributo `CLASS` para poder definir el estilo y su salida respectiva. También es posible añadir el atributo `NAME` para que algunas secciones puedan controlar el aspecto de los controles.

3. **DEFINE**: permite definir nombres de grupo y elementos correspondientes a la sección `<APP>` con más detalles a través de la etiqueta `<PROPERTIES>`.

- Sección <APP>

```
<GROUP CLASS="Dialog" NAME="Print FinishedDialog">
```

Mediante la etiqueta <**GROUP**> es posible agrupar los diferentes elementos que pertenecen a la interfaz.

El atributo **CLASS** se emplea para definir una “clase” de la interfaz de usuario de los determinados objetos a los que pertenece. La clase puede tomar como valor cualquier palabra que sea de tipo alfanumérica, siempre y cuando sea única.

Mediante el atributo **NAME** es posible asignar un nombre. El diseñador será el encargado de llevar a cabo esta tarea, de la misma forma que con la clase.

Existe la posibilidad, además, de crear una serie de botones que lleven a cabo alguna función específica. En este caso, es posible asignar a todos el mismo nombre de la clase pero con distintos valores para su atributo **NAME**.

```
<ELEM CLASS="DialogMessage" NAME="PrintFinishedMsg"/>
```

Permite definir un elemento *hoja* (que no tiene ningún hijo).

<**ELEM**> no es la etiqueta específica para cierre, pero sí que puede determinar el cierre en el caso en el que se cumplan las restricciones propias del lenguaje XML.

```
<DEFINE NAME="OKButton">
```

Ofrece la posibilidad de especificar con detalle las principales características que tiene un objeto de la interfaz, determinado por **NAME**.

```
<PROPERTIES>
```

Define las respuestas de la interfaz a las distintas acciones del usuario.

```
<ACTION VALUE="PrintFinishedDialog.Existes=False" TRIGGER="select" />
```

Especifica aquellas acciones que pueden desarrollarse por el elemento y se define dicha acción.

La sintaxis para acceder a un atributo es:

```
Elemento.Atributo=valor
```

Siendo los atributos que se pueden modificar:

- Visible
- Exists
- Enabled

- **Estilo**

Los estilos definidos en UIML pueden ser heredados de padres a hijos. Hay dos elementos que se definen cuando se aplican estilos:

1. **Elementos Toolkit:** declaran el conjunto de los elementos que se pueden utilizar en la interfaz procedentes de la librería que se utilice.
2. **Rendering:** indica el tipo de elemento que se trata. En algunas ocasiones puede ir acompañado de *RENDERING-PREFIX*, que define un prefijo que se corresponde con el nombre del paquete que contiene el elemento UI.

- **Cuadro de diálogo**

Tienen un comportamiento muy parecido a los lenguajes definidos anteriormente.

Para crear una interfaz modal en **UIML**, se muestra a continuación un ejemplo:

```
<UIML>
  <HEAD>
    <AUTHOR>Stephen King</AUTHOR>
    <DATE>October 14, 2013</DATE>
    <VERSION>1.0</VERSION>
  </HEAD>
  <APP CLASS "App">
    <GROUP CLASS="Dialog"
NAME="PrintFinishedDialog">
      ELEM CLASS="DialogButton"
NAME="OKButton"/>
    </GROUP>
  </APP>
  <DEFINE NAME="OKButton">
    <PROPERTIES>
      <ACTION
VALUE="PrintFinishedDialog.EXISTS=false"/>
    </PROPERTIES>
  </DEFINE>
</UIML>
```

2.6. MXML (*Macromedia eXtensible Markup Language*)

Este lenguaje se suele utilizar unido a aplicaciones **Adobe Flex** para llevar a cabo el diseño de interfaces. Cuenta con una estructura jerárquica, en la que el elemento raíz es el *Application* y, a partir de esta, cuelgan el resto de elementos de la interfaz.

Cuando se compila un archivo **MXML**, se obtiene como resultado un fichero *.swf.

- **Sintaxis**

En *MXML* se describen los distintos elementos mediante etiquetas que deben comenzar por **mx**. Estas etiquetas tienen un cierre u otro, dependiendo de si son *nodos hoja* o no.

- **Si son nodos hoja** se representan mediante `</>`.
- **Si no son nodos hoja** cuentan con una etiqueta para la apertura y otra para cierre.

Es posible utilizar los atributos que contienen la etiqueta para llevar a cabo su apariencia y un comportamiento determinado.

Se establece la posición de los distintos elementos utilizando los atributos **x** e **y**. Cuando se desee establecer una posición determinada, es posible hacerlo de la siguiente forma:

`<mx:Label text="Ejemplo de mxml" x=20 y=30/>`

- **Controles**

Cuenta con bastantes posibilidades para los diferentes controles en el diseño de interfaces, como pueden ser:

Button	ComboBox	Label	Image
CheckBox	DataGrid	LinkButton	HSlider
ColorPicker	DateField	List	HorizontalList

Estos controles actúan como un contenedor de otros controles MXML:

Canvas	ControlBar	Form	FormHeading
Grid	HBox	HDrivenBox	

2.7. Herramientas libres y propietarias para la creación de interfaces de usuario multiplataforma. Herramientas para crear interfaces, editor XML

A continuación, se enumera un listado con las características de cada lenguaje:

- **Netbeans:** entorno gráfico para la implementación de varios lenguajes, incluido el XML. Sus características se han mencionado anteriormente.
- **Visual Studio:** herramienta propietaria de *Microsoft* y orientada a la implementación de código en lenguajes basados en dicha compañía (por ejemplo, C#), además de lenguajes tipificados y usados con normalidad, como puede ser XML. También ha sido mencionado en apartados anteriores.
- **SharpDevelop:** es una herramienta de libre código, orientada a la programación .NET y una buena alternativa al Visual Studio pero, en este caso, con licencia GPL.

Se puede acceder a:

<http://www.icsharpcode.net/OpenSource/SD/Default.aspx>

para su descarga o más información.

- **Xamarin Studio:** entorno libre y gratuito adaptado en el anterior, pero para plataformas *Linux*.

<http://www.monodevelop.com/>

- **Gtk+:** conjunto de librerías multiplataforma para desarrollar entornos gráficos. Tiene código abierto, ya que está orientado para ser utilizado en *Linux*, aunque actualmente se puede hacer uso de él desde cualquier plataforma (*Windows* o *IOS*).

Incluso, en *Linux*, el entorno *Gnomen* ya incluye una herramienta que utiliza estas librerías y se denomina *Glade*.

<https://glade.gnome.org/>

2.8. Edición del documento XML

Un documento XML tiene dos estructuras: la parte **lógica** y la parte **física**.

Es un documento compuesto por entidades identificadas por un nombre. Se compone de elementos, atributos y valores. Cuando se finaliza la implementación del fichero, es preciso analizarlo mediante una herramienta definida para tal fin, que se denomina **Parser**.

De esta forma se comprueba si el documento es válido. Un analizador tiene como objetivo transformar el código XML en código legible.

Existe una gran variedad de analizadores XML, como, por ejemplo, **xerces**, que es de código libre y multiplataforma:

<https://xerces.apache.org/xerces-c/>



Para escribir cualquier código XML es posible utilizar un editor cualquiera (bloc de notas, por ejemplo), aunque es preciso señalar que existen editores que facilitan la implementación de código sobresaltando las marcas claves con distintos colores y técnicas para visualizar en un golpe de vista las partes que compone el fichero.

3. Creación de componentes visuales

3.1. Concepto de componente y características

Un componente es un elemento que cuenta con suficiente autonomía y funcionalidad para existir por sí solo, pudiendo adaptarse a diferentes situaciones.

Gracias al diseño de componentes, es posible definir una estructura de diseño modular que no tenga demasiado acoplamiento y se mantenga unida mediante la programación modular.

Es bastante frecuente encontrar en Internet pequeños componentes que ofrecen distintas posibilidades a la hora de resolver una determinada función cotidiana que puede tener más o menos complejidad. Algunos ejemplos de ello son el conversor de monedas, las conversiones de archivos, entre otros.

A la manera de unir distintos componentes y desarrollar un determinado código para conseguir un correcto funcionamiento, se le denomina **desarrollo de software basado en componentes** y, entre sus principales ventajas, destacan:

- Reutiliza software.
- Permite simplificar las pruebas.
- Permite simplificar el mantenimiento del sistema.
- Mayor calidad.

Una alternativa al desarrollo del código sería comprarlo a terceros que lo tengan ya implementado.

En este caso, las principales ventajas que se aprecian son:

- Posee ciclos de desarrollo más cortos.
- Mejor ROI.
- Consigue mejorar la funcionalidad.

3.2. Propiedades y atributos

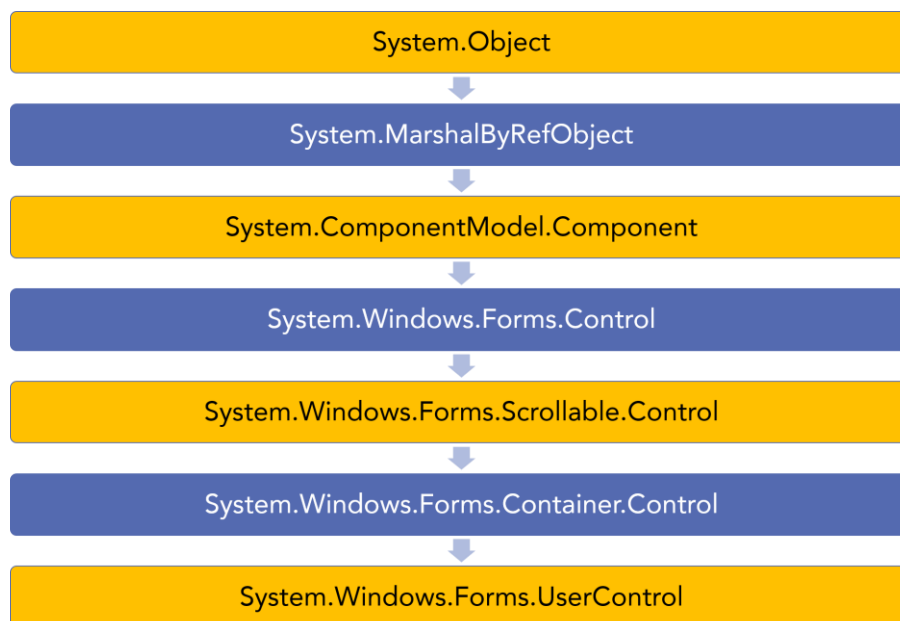
En este apartado se hace referencia a los diferentes controles predefinidos en **Visual Studio**, y los más comunes a la hora de diseñar interfaces, son, entre otros, las **etiquetas, botones y desplegables**.

Los diferentes controles que se pueden seleccionar vienen definidos en una lista denominada caja de herramientas que puede situarse en cualquier lugar.

A la hora de buscar un elemento que no pertenece a esa lista, es posible buscarlo en otras opciones del menú.

En cualquiera de los casos, aparecerá una ventana con una serie de controles disponibles en las librerías que se referencian en un determinado proyecto.

En las librerías **WPF** los distintos componentes se crean a partir de la clase **UserControl**, y permiten heredar todas sus propiedades, métodos y eventos.



Es fundamental elegir la clase correcta, es decir, aquella que más se acerque a las necesidades del usuario. De esta manera, se ayuda al desarrollador en el tiempo de implementación y en el diseño.

3.3. Elementos generales

A continuación, se desarrollan una serie de elementos que permiten una mejor lectura y acceso a las distintas propiedades de los elementos existentes:

- **Button:** permite realizar una acción u otra en exclusividad. Los distintos atributos que se pueden utilizar son:
 - **Content:** selecciona el texto que se va a mostrar en el botón.
 - **InCancel:** ofrece la posibilidad de asociar el botón a una acción determinada al pulsar la tecla *esc* del teclado.
 - **InDefault:** ofrece la posibilidad de asociar el botón a una acción determinada al pulsar la tecla *INTRO* del teclado.
 - **DataContext:** se puede utilizar para unificar una serie de controles que se encuentran dentro de un control contenedor a un origen de datos.
 - **IsEnabled:** especifica si el control está activo o no ante las diferentes acciones que el usuario desee realizar.
 - **ToolTip:** trozo de texto no muy grande que se puede visualizar cuando se posiciona el puntero del ratón sobre el control.

- **TextBox:** caja de texto que ofrece la posibilidad de realizar una determinada operación en la que el usuario actúa de forma interactiva a la hora de introducir algún tipo de información. Entre sus atributos destacan:
 - **Text:** especifica el texto que se introduce en cada elemento.
 - **UndoLimit:** especifica la cantidad de operaciones que se pueden realizar.

- **ListBox:** lista con una serie de elementos que pueden ser seleccionados por el usuario. Sus atributos principales son:
 - **Items:** se refiere al conjunto de elementos que forman parte de la lista.
 - **ItemsSource:** hace referencia al origen de los datos que van a determinar los elementos de una lista.
 - **SelectedIndex:** selecciona un elemento de una lista, asignando el valor de “-1” si no existe el elemento.

- **ComboBox:** unión de una caja de texto a una lista con sus propiedades correspondientes y ventajas. Entre sus atributos se encuentran:
 - **IsDropDownOpen:** permite seleccionar el elemento de la lista.
 - **IsEditable:** determina si la caja de texto es editable y, si es así, es posible introducir texto.
 - **IsReadyOnly:** no permite realizar ninguna modificación sobre los elementos ya que es un elemento de solo lectura.

- **CheckBox:** determina el elemento que se va a utilizar para definir las diferentes opciones. Es posible seleccionar más de una opción. Atributos principales:
 - **ClickMode:** especifica si es posible liberar el evento click, diferenciando las opciones.
 - **Release:** libera la pulsación.
 - **Press:** tiene lugar la pulsación.
 - **Hover:** pasa por encima del elemento.
 - **Content:** determina el texto que está asociado al elemento.
 - **IsChecked:** especifica el estado que tiene un elemento por defecto, es decir, si está seleccionado o no.
 - **IsThreeState:** especifica los tres estados diferentes que puede tener el control.

- **RadioButton:** es un elemento muy similar al anterior, en este caso, las distintas selecciones son excluyentes unas de otras.

- **Image:** este control permite añadir imágenes en una interfaz haciendo referencia a los diferentes recursos. Entre sus principales atributos se señalan:
 - **Source:** ruta que representa el archivo que se muestra.
 - **Stretch:** permite modificar el tamaño de una imagen al espacio disponible de la aplicación.
 - **StretchDirection:** especifica la forma en la que se lleva a cabo el ajuste.

3.3.1. Creación básica de un componente WPF

A la hora de crear un nuevo componente gráfico WPF, se ofrece al desarrollador la opción de partir de la clase *UserControl* o reutilizar algún control o incluso varios:

- **Control personalizado:** a modo de ejemplo, es posible iniciar partiendo de un control que no sea muy complicado, como puede ser la caja de texto.

La función principal de una caja de texto permite solo valores numéricos y, si se introduce cualquier otro dato, avisará con un mensaje de error. Pasos para solucionarlo:

- Iniciar con el control propio *TextBox* y extender esta clase añadiendo la función que se desee.
 - Añadir la librería *PresentationFramework* dentro del proyecto.
- **Definición de propiedades:** las propiedades asociadas a los componentes gráficos no son muy complejas de definir, solo es necesario utilizar aquellas propiedades determinadas de una clase para hacer uso de sus comportamientos.

Estas propiedades pueden ser tanto **simples** como **indexadas**.

- **Testing del componente:** a la hora de realizar las pruebas de *testing* de un determinado componente es necesario otro proyecto que se pueda ejecutar por sí mismo.

Los componentes solo funcionan si se encuentran dentro de un proyecto ejecutable como, por ejemplo, la aplicación de un formulario.

- **Métodos:** crear diferentes métodos para componentes gráficos es muy similar a crear una determinada función perteneciente a una clase que defina al componente. La función en cuestión es la que va a determinar la tarea que debe llevar a cabo.

3.4. Eventos; asociación de acciones a eventos

Eventos. Se refieren a una acción determinada que puede desempeñar un usuario. Esta acción está asociada a un componente específico.

Cuando se realiza un evento se producen una serie de acciones, como, por ejemplo, la pulsación de un botón o la salida de un campo de texto.

- **Creación de un evento**

Un evento es la función que se desencadena tras haber realizado una acción determinada sobre un componente en cuestión.

El principal elemento de un evento se denomina **manejador** (*handler*) y se especifica mediante la siguiente función:

```
This.TextChanged+=new  
System.Windows.Controls.TextChangedEventHandler(manejador);
```

Donde el manejador corresponde a una rutina que tiene una **firma** (*signature*) que cumple todos los requisitos que tiene asignados.

- **Diseño de eventos**

Existen dos maneras diferentes para diseñar eventos:

- Hacer uso de un evento ya definido y modificarlo hasta conseguir adaptarlo a las necesidades del usuario.
- Crear un evento desde cero partiendo de un nuevo componente.

Es posible realizar el manejador en la clase parcial de código que esté asociada al archivo XAML. El manejador es un método que contiene una subrutina (pública o privada) en la clase. Esta subrutina cuenta con dos parámetros de entrada:

1. **Sender:** permite representar el objeto asociado al manejador.
2. El segundo cuenta con información específica sobre un determinado evento.

- **Eventos enrutados**

Gracias a los eventos enrutados es posible pasar un evento desde un hijo hasta sus correspondientes padres, igual que una estructura árbol.

Cuando se propaga un evento mediante una ruta ascendente, los manejadores que estén asociados al evento pueden compartir la instancia de datos del evento, por tanto, si se realiza cualquier modificación, esta se va a propagar a partir de ese punto. En este caso, el objeto **sender** va a cambiar, pasando a ser el elemento que esté asociado al manejador.

- **Escuchadores (*Listener*)**

Los escuchadores son los encargados de controlar los eventos, y, para ello, deben esperar a que el evento se produzca.

Dependiendo del evento que se produzca, será necesario un escuchador u otro, ya que los escuchadores están formados por una serie de métodos que se deben implementar, aunque solo se utilice uno.

Los escuchadores se encuentran en ***java.awt.event***.

3.4.1. Asociación de acciones a eventos

MÉTODOS	Public void actionPerformed (ActionEvent)	
EVENTOS	JButton	Click o pulsar INTRO cuando el foco está acivado en él
	JList	Doble click en un elemento de una lista
	JMenuItem	Selecciona una opción del menú
	TextField	Al pulsar INTRO con el foco activado
ESCUCHADOR	ACTIONLISTENER→Click sobre el componente o si se hace INTRO cuando este tiene el foco	

MÉTODOS	public void keyTyped (KeyEvent e)	
	public void keyPressed (KeyEvent e)	
	public void keyReleased (KeyEvent e)	
EVENTOS	keyTyped	Al pulsar y soltar la tecla
	keyPressed	Al pulsar la tecla
	KeyReleased	Al soltar la tecla
ESCUCHADOR	KEYLISTENER→cuando se pulsa una tecla, según el método, cambia la forma	

MÉTODOS	public void focusGained (FocusEvent e)	
	public void focusLost (FocusEvent e)	
EVENTOS	LostFocus	Cuando pierde el foco.
ESCUCHADOR	FOCUSLISTENER→cuando un componente gana o pierde el foco (el que está seleccionado)	

MÉTODOS	Public void mouseClicked(MouseEvent e)	
	Public void mouseEntered(MouseEvent e)	
	Public void mouseExited(MouseEvent e)	
	Public void mousePressed(MouseEvent e)	
EVENTOS	mouseClicked	Pinchar y soltar
	mouseEntered	Entra en un componente con el puntero
	mouseExited	Sale de un componente con el puntero
	mousePressed	Presiona el botón
	mouseReleased	Suelta el botón
ESCUCHADOR	MOUSELISTENER→cuando se lleva a cabo alguna acción con el ratón.	

MÉTODOS	Public void mouseDragged(MouseEvent e)	
	Public void mouseMoved(MouseEvent e)	
EVENTOS	mouseDragged	Click y arrastrar un componente
	mouseMoved	Cuando se mueve un puntero sobre un elemento
ESCUCHADOR	MOUSEMOTIONLISTENER→se produce con el movimiento del ratón.	

4. Usabilidad

Interfaz gráfica de usuario

La interfaz gráfica de usuario (**GUI**) hace referencia a cómo los distintos elementos gráficos permiten comunicarse con un determinado sistema informático.

Aunque se hable de interfaz gráfica, se hace referencia no solo a ordenadores, sino también a dispositivos móviles, tablets, monitores táctiles, etc.

A continuación, se detallan algunos de sus principales objetivos:

- **Funcionalidad:** se encarga de facilitar las tareas que se llevan a cabo.
- **Amistosa:** cuenta con un manejo bastante sencillo.

Es conveniente señalar que, en ocasiones, las interfaces son importantes porque existen casos concretos en los que el usuario debe realizar una tarea concreta y no sabe cómo hacerlo o a dónde se tiene que dirigir, por lo que se recomienda siempre que tenga una apariencia amigable y fácil de manejar. De esta forma, será más sencillo poder guiar al usuario por las distintas opciones que necesite.

Diseño de una interfaz

A la hora de llevar a cabo el diseño de una interfaz, es necesario basarse en una serie de principios básicos que es conveniente tener en cuenta:

- **Sencilla:** debe disponer de una serie de elementos que sean capaces de ayudar y guiar. Hay que evitar la existencia de interfaces que estén muy cargadas.
- **Clara:** es importante que la información se pueda visualizar de manera fácil siguiendo un criterio de organización sencillo, lógico y jerárquico.
- **Predecible:** en caso de existir las mismas funciones, deberán tener las mismas respuestas.
- **Flexible:** debe contar con una interfaz homogénea para poder utilizarla en distintas plataformas.
- **Consistente:** una vez organizados los elementos, es conveniente que permanezcan en la misma área. La página que más variaciones puede desarrollar es la principal, aunque las funciones permanecerán igual.
- **Intuitiva:** si la interfaz se ha desarrollado de una forma sencilla, el usuario debe poder acceder a cualquier función sin ninguna dificultad.
- **Coherente:** es preciso conseguir que sea más entendible si se añaden palabras o frases a los distintos elementos, como pueden ser, entre otros, gráficos y colores.

4.1. Concepto de usabilidad

“Podemos definir la usabilidad como la medida en la cual un producto puede ser usado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado”.

Diario Accesibilidad Universal

La usabilidad está basada en ofrecer distintas situaciones, tan simples como que, un determinado usuario, cuando visualice una interfaz, pueda ser capaz de encontrar la información que pretende.

La usabilidad de una interfaz puede ser considerada como una medida de utilidad, de fácil uso y aprendizaje para una tarea, usuario y un determinado contexto.

- **Facilidad de aprendizaje:** los usuarios nuevos deben poder interactuar de forma efectiva con el sistema.
- **Facilidad de uso:** el usuario debe poder hacer uso de la herramienta de forma sencilla y utilizando el menor número de pasos posibles para conseguir llegar a un sitio determinado.
- **Flexibilidad:** tanto el usuario como el sistema deben poder interactuar e intercambiar información.
- **Robustez:** grado de ayuda con la que cuenta el usuario para realizar las diferentes acciones en el programa gráfico.

Los atributos que más se suelen utilizar en una interfaz son, entre otros:

- Tiempo de aprendizaje.
- Eficiencia de uso.
- Retención a través del tiempo.
- Confiabilidad en el uso.
- Satisfacción a largo plazo.
- Eficacia.
- Entendimiento.
- Adaptabilidad.
- Primera impresión.
- Elementos extra.
- Desempeño inicial.
- Evolucionable.

Problemas detectados

Jakob Nielsen, una de las personas más respetadas en el mundo por su estudio sobre la usabilidad, estableció los diferentes problemas que podía plantear la usabilidad para, a partir de ellos, diseñar una serie de reglas que permitieran identificar estos posibles errores.

- **Visibilidad del estado del sistema:** para poder informar a los diferentes usuarios, el sistema siempre debe permanecer abierto haciendo uso de la retroalimentación.
- **Relación entre el sistema y el mundo real:** para que el sistema se pueda comunicar con el mundo real, estos deben hablar el mismo idioma. Por tanto, será preciso unificar la información con la que se cuenta para conseguir establecer esta comunicación de forma natural.
- **Control y libertad de usuario:** son muy prácticas las funciones de hacer y resolver para poder encontrar una “salida” en caso de que se produzca algún error.
- **Consistencia y estándares:** no es tarea de los usuarios cuestionar si las acciones y las palabras diferentes significan lo mismo.
- **Prevención de errores:** es más importante prevenir los errores que diseñar unos buenos mensajes de error.
- **Reconocimiento antes que recuerdo:** tanto los objetos como las acciones y las opciones deben ser visibles. No es tarea del usuario el recordar aquella información que le permite continuar. Para ello, debe contar con instrucciones visibles.
- **Flexibilidad y eficiencia de uso:** mediante el teclado y combinando distintas teclas, es posible interactuar de forma más rápida.
- **Estética y diseño minimalista:** los diálogos no deben contar con información que no sea importante.
- **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores:** los distintos mensajes de error deben ser entregados en un lenguaje conciso y claro.
- **Ayuda y documentación:** es recomendable contar con una serie de ayuda a la que poder recurrir en caso de error. Esta información no debe ser muy complicada y debe ser de fácil acceso.

Beneficios de la usabilidad

A continuación, se detallan, entre otros, los principales beneficios de la usabilidad:

- Reduce los costes de aprendizaje y esfuerzo.
- Disminuye los costes de asistencia y ayuda al usuario.
- Disminuye la tasa de errores provocados por el usuario.
- Optimiza los costes de diseño, rediseño y mantenimiento.
- Aumenta la tasa de conversión de visitantes a clientes de un sitio web.
- Aumenta la satisfacción y comodidad del usuario.
- Mejora la imagen y prestigio.
- Mejora la calidad de vida de los diferentes usuarios.

Estándares de la usabilidad

Se pueden encontrar una gran cantidad de estándares relativos a la usabilidad (**estándar ISO 9241**). Esta norma está centrada en la calidad de la usabilidad y ergonomía, tanto de hardware como de software. Creada por ISO y la IEC y actualizada y mejorada hasta ISO/IEC 9241-9:2001.

Este estándar permite ofrecer una serie de requisitos que están relacionados con los atributos del hardware, software y el entorno, y permiten atribuir la usabilidad junto a determinados principios ergonómicos.

A continuación, se enumeran las diferentes estandarizaciones creadas hasta la fecha relativas a la usabilidad:

- **ISO/IEC TR9126**: define la usabilidad en términos de claridad, facilidad de aprendizaje, operatividad y atractivo. En las partes 2 y 3 se definen las métricas.
- **ISO/IEC TR9126-2**: Ingeniería de software - Calidad de producto - Parte 2 Indicadores externos.
- **ISO/IEC TR9126-3**: Ingeniería de software – Calidad de producto – Parte 3 indicadores externos.
- **ISO 9421**: requisitos ergonómicos para trabajos de oficina con pantallas de visualización.
- **ISO 11064**: diseño ergonómico de centros de control.
- **ISO 141915**: la ergonomía de software para interfaces de usuario multimedia.
- **IEC TR 61997**: directrices para las interfaces de usuario en equipos multimedia para uso general.

- **ISO/IEC 10741-1:** Interacción Diálogo-Control del cursor para editar texto.
- **ISO/IEC 11581:** Icon Symbols and functions.
- **ISO 13406:** requisitos ergonómicos para trabajos con pantallas visuales basadas en pantallas planas.
- **ISO/IEC 14754:** Interfaces basadas en lápiz – Gestos comunes de edición de texto con los sistemas basados en el lápiz.
- **ISO/IEC 18021:** Tecnología de la información – Interfaz de usuario para las herramientas móviles.
- **ISO 18789:** requisitos ergonómicos y técnicas de medición para pantallas visuales electrónicas.

4.2. Medidas de usabilidad

La función principal de las medidas de usabilidad es valorar el grado al que han conseguido llegar tanto el usuario como las organizaciones para que, de esta forma, se pueda proporcionar una información para que se pueda utilizar por los diseñadores y desarrolladores con la intención de mejorar el estado de la interfaz.

La cantidad de métodos de evaluación puede variar según su grado de formalidad y participación del usuario.

El método más adecuado dependerá del producto evaluado, de la disponibilidad que tengan los usuarios más representativos, y de las restricciones que tenga añadidas.

A la hora de llevar a cabo la evaluación se basa en:

- El **usuario**: ofrece información relacionada con la tarea que se va a realizar.
- El **experto**: describe la falta de conformidad que existe con las normas o directrices de diseño de la interfaz.

Se pueden diferenciar entre **dos tipos de objetivos principales** para llevar a cabo el análisis y la medición:

1. Diagnóstico de problemas de usabilidad

- Aquellos métodos que están basados en el usuario, como:
 - Evaluación participativa.
 - Evaluación de diagnóstico.
 - Análisis de incidentes críticos.
- Aquellos que se pueden completar por el experto o por la evaluación heurística.

2.Evaluación para comprobar si se han conseguido o no los objetivos referentes a la usabilidad

- Los requisitos para satisfacción y desarrollo del usuario se pueden evaluar mediante la utilización de pruebas de rendimiento, carga de trabajo cognitivo, etc.
- Otros objetivos diferentes de usabilidad se puedan evaluar a través de la evaluación de expertos.

Es conveniente que todos estos métodos se utilicen para probar los diferentes diseños finales. Además, los métodos pueden ofrecer una gran cantidad de información de diagnóstico que puede ser utilizada para mejorar o añadir requisitos en siguientes versiones.

Las herramientas más utilizadas para la evaluación suelen ser, en muchos de los casos, las **entrevistas**, **test** y **cuestionarios**.

Tipo de métrica

- **Directa**

La métrica directa hace referencia a la relación que va desde un determinado atributo hasta un número, pudiendo servir como referencia para realizar la evaluación y la descripción de diferentes situaciones del mundo real.

Normalmente, los atributos se miden a través de métricas directas.

A continuación, se muestran algunos ejemplos:

- **Longitud del texto del cuerpo de una página:** medido por la cantidad de palabras.
- **Cantidad de enlaces rotos internos de un sitio web:** medido según la presencia de errores del tipo 404.
- **Cantidad máxima de *frames* que tiene un sitio web:** medidos por la presencia de etiquetas FRAMESET.
- **Cantidad de imágenes contexto alternativo de un sitio web:** medido por las etiquetas *ALT* que existen en cada una de las imágenes que están vinculadas al sitio web.

- **Indirecta**

La métrica indirecta se refiere a la relación que existe entre dos o más atributos. Normalmente, las características y subcaracterísticas se miden mediante métricas indirectas.

Por ejemplo:

- Porcentaje de los enlaces rotos de un determinado sitio

$$(N^{\circ} \text{ enlaces rotos internos} + N^{\circ} \text{ enlaces rotos externos}) * 100 / N^{\circ} \text{ total enlaces}$$

- Porcentaje de la presencia de la propiedad ALT

$$(N^{\circ} \text{ imágenes ALT} / N^{\circ} \text{ total imágenes}) * 100$$

- **Interna**

Se refiere a un valor numérico del atributo que involucra al valor en sí, independientemente de si es obtenido por una métrica directa o indirecta.

- **Externa**

Se refiere a un valor resultante del atributo cuando se aplica una métrica indirecta. En este caso, involucra al sujeto junto al comportamiento con el entorno.

- **Objetiva**

Se refiere a un valor resultante del atributo de un determinado sujeto. En este tipo de métrica, es posible distinguir entre diferentes grados de objetividad.

- **Subjetivas**

Por último, la métrica subjetiva se refiere a un valor numérico que siempre involucra al usuario mediante heurísticas o distintos criterios de preferencia.

4.3. Pautas de diseño de interfaces

Existen diferentes pautas sobre el buen funcionamiento del diseño de una interfaz, de tal manera que es posible obtener un determinado producto que ofrezca al usuario la posibilidad de interactuar correctamente en la aplicación.

Un diseño preciso permite que la apreciación que se ha obtenido por el usuario sea la de estar ante una buena interfaz, independientemente de si existen o no elementos gráficos muy llamativos.

4.3.1. Pauta de la estructura

Es muy importante realizar un diseño correcto de la estructura de una interfaz para conseguir un manejo fácil y comprensible para el usuario.

Existen una serie de recomendaciones que se pueden verificar de forma gráfica:

- **Jerarquía:** clasificación del orden de los elementos principales.
- **Foco:** determina una zona de bastante importancia. Se sitúa en la esquina superior izquierda.
- **Homogeneidad:** la estructura de las diferentes ventanas debe ser uniforme para conseguir una mejor interpretación de la interfaz.
- **Relaciones entre elementos:** es conveniente organizar los elementos que estén relacionados en la misma zona de forma limpia y concisa. De esta manera, se facilitará la lectura de la interfaz.

Para alinear los diferentes elementos se siguen una serie de **reglas**:

- Si los textos tienen la misma longitud, es preciso alinear los cuadros de texto a la izquierda.
- Si existe un conjunto de etiquetas (textos) de diferentes longitudes, hay que alinear los textos a la derecha.
- Si existe un conjunto de etiquetas (textos) con las mismas longitudes, se alinean los textos a la izquierda.
- La distancia que hay entre la etiqueta y el elemento no debe ser demasiado grande, para conseguir que sea más fácil la relación entre ellos.
- Es conveniente agrupar todos los elementos mediante la utilización de los encabezados y espaciados. Estos espaciados permiten organizar mentalmente

cómo va a ser la interfaz para, de esta manera, conseguir mejorar su interactividad.

- Los criterios de tamaño y alineación de los diferentes elementos deben ser los mismos para todas las ventanas, así se consigue hacer más fácil su lectura y comprensión.
- Por último, es recomendable que el usuario no acceda a varias zonas de la aplicación de forma muy rápida ya que puede provocar un efecto de mareo.

Acceso rápido. Hacer uso de las diferentes combinaciones de teclas para conseguir un acceso más rápido a las opciones de la interfaz. Se pueden considerar estas combinaciones como un “atajo” que llevan a un sitio, pero de forma más directa.

4.3.2. Cuadro de diálogo

Es conveniente que los cuadros de diálogo sean lo más sencillos posibles, ya que su función principal es simplificar la resolución de una función que se encuentra dentro de la interfaz principal.

4.3.3. Desplegables

Es conveniente hacer uso de las listas desplegables ya que ofrecen la posibilidad de poder seleccionar uno o algunos de los elementos, según las necesidades que existan en cada momento.

- **Flujo de ejecución:** ofrece al usuario, de forma clara, el flujo de ejecución que puede existir entre las diferentes ventanas. Esto es posible estableciendo en varias aplicaciones la utilización de *TabOrder*, que informa sobre cómo se va a mover el cursor cada vez que se pulsa la tecla *Tab*.
- **Integración:** permite crear diferentes elementos gráficos en lugares en los que se va a utilizar.

4.3.4. Fuentes

Según las fuentes utilizadas, se pueden provocar una serie de reacciones diferentes en los usuarios, como las indicadas a continuación:

- Es conveniente utilizar una **semántica clara** para el texto. Si se van a referenciar elementos iguales, es aconsejable la utilización de la misma semántica y así dejar constancia de que se va a realizar la misma acción, evitando la ambigüedad.

- No es conveniente emplear la misma **etiqueta** en un mismo formulario.
- Es preciso tener **estandarizado el uso de fuentes**, ya que, si se emplean muchos tipos distintos, se dificulta la lectura al usuario.
- Se pueden utilizar **gráficos**, pero no es conveniente establecerlos como fondo de pantalla, ya que dificultan bastante la visión.
- No es conveniente redactar texto con todos los caracteres en **mayúscula**. Es preferible reservar este tipo de letras para un uso más limitado como pueden ser los encabezados.
- Se debe utilizar un **lenguaje** que sea apto para todo el público, buscando los conceptos clave como los más significativos y evitando el uso de palabras negativas como error, fallo, etc. Lo ideal es hacer uso del mismo tipo de mensajes para las mismas acciones.
- En caso de que se produzca algún **error**, es preferible resaltar cómo se va a solucionar, en lugar de a qué corresponde ese tipo de error.

4.3.5. Colores

El uso de los diferentes colores puede provocar una mejor percepción de la interfaz, ya que permite resaltar aquellas tareas principales. Aunque, un abuso de ellos, produce un efecto rebote, haciendo que sea más complicada la lectura de la interfaz.

Como en apartados anteriores, se pueden seguir una serie de **reglas** que aclaren los factores más importantes que se deben tener en cuenta a la hora de seleccionar un color o combinarlo.

- No es recomendable utilizar más de cuatro colores en una misma ventana. El número seleccionado no debe superar más de siete en toda la aplicación.
- Se debe hacer un uso de los colores de forma uniforme y utilizar un mismo color para las mismas funciones.
- Es posible modificar el color para indicar el estado de un usuario. Por ejemplo, color rojo si hay algún error, o verde si el proceso es correcto.
- También se puede utilizar un determinado color para destacar las zonas principales de la interfaz.
- Es conveniente no utilizar combinaciones de colores que dificulten la lectura.

4.3.6. Gráficos

Mediante la utilización de gráficos es posible facilitar una lectura rápida a los usuarios. Estos elementos ocupan más espacio que el texto, por lo que pueden provocar una sobrecarga mayor.

El uso de los iconos es más extendido, sobre todo para los entornos móviles en los que el espacio es fundamental. Es conveniente que los iconos estén diseñados de forma correcta para que se pueda identificar su función de manera rápida.

Las principales características que se deben tener en cuenta a la hora de desarrollar interfaces son las siguientes:

- Deben conseguir **captar la atención del usuario**.
- **Optimizar el espacio**: transmiten una determinada función sin llegar a especificarlo mediante un texto.

4.4. W3C

En este apartado se tienen en cuenta la web que ya se encuentra diseñada, para que todos los usuarios puedan interactuar con ella, independientemente del software que se utilice, del lenguaje, la localización y la habilidad, tanto física como mental que se pueda tener.



Una determinada web tiene como objetivo principal ser accesible para el mayor número de usuarios posibles.

Según **Tim Berners-Lee**, W3C Director e inventor de *www*,

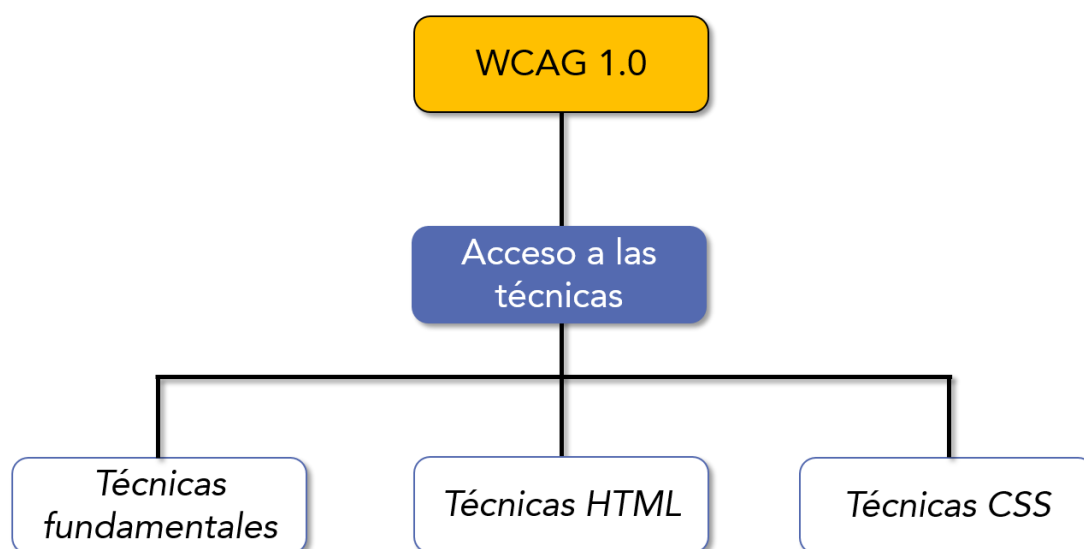
“El poder de la web radica en su universalidad. Su acceso universal a pesar de las diferentes discapacidades es un elemento esencial”.

Es fundamental el impacto visual que presente una aplicación web, permitiendo la eliminación de barreras de acceso tanto en la comunicación como en la interacción

que pueden sufrir los diferentes usuarios. Si bien es cierto que existen muchos diseños web que impiden su uso a algunos colectivos.

Uno de los principales objetivos de la accesibilidad web es poder dirigirla hacia un uso de accesibilidad para, de esta manera, poder conseguir que aquellas personas que presentan algún tipo de discapacidad, puedan participar en la web como todas las demás.

Las **WCAG 1.0** disponen de 14 pautas que son necesarias para conseguir un diseño accesible al alcance de todos los usuarios. Cada una de estas pautas cuenta con una serie de puntos para determinar las distintas áreas.



Cada pauta **WCAG 2.0** desarrolla diferentes criterios de éxito, existiendo hasta 60 criterios o distintos puntos de verificación o comprobación que van a permitir determinar el nivel de accesibilidad (A, AA, AAA).

Estos criterios de éxito están ordenados según su nivel de cumplimiento (A, AA, AAA). Para que cada página web cumpla con estas **Pautas WCAG 2.0**, tiene que cumplir una serie de requisitos de conformidad:

- **Nivel de conformidad “A”:** se satisfacen todos los puntos de verificación correspondientes a la prioridad 1.
- **Nivel de conformidad “AA” (Doble A):** se satisfacen todos los puntos de verificación correspondientes a las prioridades 1 y 2.
- **Nivel de conformidad “AAA” (Triple A):** se satisfacen todos los puntos de verificación correspondientes a las prioridades 1, 2 y 3.

Aquellas páginas que cumplen las Pautas WCAG 2.0 reciben una **Declaración** en la que se referencia a los diferentes usuarios que cumple con el **W3C**.

Es posible representar el nivel de conformidad mediante los siguientes logotipos:



Cuando se hace uso de alguno de estos sellos es preciso acompañarlos de la siguiente información:

- Fecha de revisión del cumplimiento.
- Título, versión y URI de las pautas WCAG 2.0.
- Nivel de conformidad alcanzado (A, AA, AAA): solo es aplicable a la página web completa. No es posible alcanzar la conformidad si se excluye alguna parte de la página web.
- Alcance: enumeración exacta de aquellas páginas que cumplen las Pautas WCAG 2.0.
- Lista de las diferentes tecnologías de las que depende el contenido.

5. Confección de informes

La creación de informes es algo bastante frecuente cuando se trata de diseño de aplicaciones, ya sean de escritorio o como web. Es muy importante mostrar a los distintos usuarios un tipo de información organizada en distintos formatos.

Esta funcionalidad aumenta según la cantidad de datos que se están manejando. De la misma forma, también se cuenta con la posibilidad de adaptar la utilización de informes a otras tareas tan sencillas, como puede ser la creación de una determinada cuenta.

5.1. Informes incrustados y no incrustados en la aplicación

A la hora de presentar un informe, existen diferentes divisiones catalogadas como fijas. De todas ellas, se puede destacar la sección que está dedicada a los datos que van a formar parte del cuerpo del informe.

Las secciones que ya están definidas son:

- Cabecera
- Cuerpo
- Pie

A partir de estas tres secciones es posible definir la estructura más básica que puede llevar a cabo un informe. En ella, la cabecera se encarga de introducir distintos datos que solo son visibles en la primera página. Seguidamente, los datos se mostrarán en el cuerpo del informe y, por último, en el pie. La función principal del pie es mostrar una serie de resúmenes o datos totales.

La estructura de un informe puede complicarse a medida que se van añadiendo datos agrupados, gráficos, subinformes, etc.

5.1.1. Instalación de reporting services

La herramienta principal con la que contar a la hora de diseñar informes, **Reporting Services**, es gratuita y ofrece una serie de plantillas determinadas para el entorno de desarrollo *Visual Studio* que siempre están disponibles si se encuentran instaladas las diferentes opciones del software *SQL Server*.

Los servidores asociados a la base de datos *SQL Server* pueden soportar hasta dos modos diferentes para autenticarse:

- **Seguridad Windows integrada:** puede hacer uso de las credenciales del usuario para autenticarse en el servidor. Las cuentas que se utilizan son cuentas locales del propio sistema operativo o, a veces, del propio dominio.
- **Modo mixto:** puede hacer uso de un determinado modo de autenticación mediante usuario y contraseña. Una de las cuentas que se suele utilizar es la cuenta *sa* (*super administrador*), que permite representar el administrador del servidor.

Una vez finalizado el proceso de instalación, es importante comprobar si se han creado dos bases de datos nuevas en el servidor de bases de datos con el prefijo *ReportServer*. Estas serán las encargadas de guardar metadatos, además de alguna configuración del servidor de informes.

5.1.2. Creación de un informe

En el programa *Visual Studio*, dentro de la categoría ***Business Intelligence***, es posible comprobar que se proporcionan una serie de plantillas que se pueden utilizar para crear los distintos proyectos que se encuentren relacionados, entre otros, con la explotación de datos, como pueden ser los informes, migración y la transformación de los datos.

Por tanto, los proyectos que se van a llevar a cabo son aquellos que se pueden identificar como *Reporting Services*.

Entre todos estos proyectos se encuentra dos plantillas, bastante parecidas, tanto en su estructura como en las funciones. Añadir estas dos plantillas tan parecidas es una idea basada en el modelo de diseño de *Microsoft*, ya que cuenta con una amplia cantidad de asistentes que van a permitir al usuario simplificar las tareas pendientes. Una de sus ventajas principales es que aparta al usuario de los detalles más complejos para conseguir que la tarea sea lo más sencilla posible.

La plantilla *Report Server Project* permite crear un proyecto sin ningún asistente, solo con aquellos archivos que sean necesarios para la solución.

Esta plantilla (sin asistente) obtiene una determinada interfaz que cuenta con la estructura del proyecto como una imagen adjunta en la que se pueden ver tres carpetas.

Se comienza usando el asistente para la creación de un proyecto *Report Server* para facilitar el proceso de aprendizaje. Se basa en la ingeniería inversa, olvidándose, por un momento, de los distintos tipos de archivos que existen.

En este proceso guiado por el asistente es fundamental la relevancia de los datos que se pueden utilizar en el informe, unido al *layout* deseado para su representación.

Selección de datos

Antes de comenzar con el diseño de un informe, es preciso identificar qué es lo que se debe representar en dicho informe, junto con los datos que van a ser precisos para tal representación.

Estos datos que se van a utilizar dentro de un determinado informe, se definen una vez que se establezca el origen de los datos. Tras esto, ya será posible seleccionarlos. Llegados a este punto es posible ver cómo se utiliza un identificador, **DsAdventure**, mediante el que se puede representar una estructura que contenga todos los datos que se seleccionen.

5.1.3. Layout

Una vez concluida la selección de los datos, llega el momento de diseñar la estructura visual junto con la forma en la que va a ir organizada la información en la misma. Se pueden diferenciar dos de las formas más frecuentes a la hora de representar los datos:

- Formato tabular
- Estructura matriz

El asistente da la posibilidad de poder seleccionar cualquiera de ellas.

Es posible organizar los datos en el informe generando distintas agrupaciones por página o grupos de datos.

El asistente que existe para la selección de datos (**Generador de consultas**), permite crear grupos de datos.

Es conveniente que la organización de los datos la lleve a cabo el entorno del servidor de la base de datos, ya que apostará por la optimización y el rendimiento.

La salida que genera el asistente a la hora de organizar los datos se limita a la generación de bandas, filas, o bloques. En cualquiera de los casos, se ofrece la posibilidad de añadir una opción de tipo de subtotal.

Para finalizar, el asistente dispone de diferentes estilos predefinidos que se pueden aplicar al informe. La elección de un estilo u otro permitirá visualizar cómo sería la salida de manera aproximada.

Una vez utilizado el asistente, generará una estructura con el aspecto determinado. Llegados a este punto, se puede proceder a ejecutarlo. Mientras el informe está ejecutándose, es posible hacer uso de las herramientas proporcionadas por los servicios *Reporting Services* para poder desarrollar distintas tareas.

5.1.4. Exportación de la información

Una de las funcionalidades más demandadas en esta herramienta es la exportación de la información a determinados formatos (*cvs* o *pdf*), sobre todo a otras bases de datos. Debido a la gran importancia de esta opción, normalmente los entornos de desarrollo y sistemas gestores de bases de datos facilitan la realización de la misma, haciendo muy intuitiva dicha opción.

5.1.5. Organización del informe

Tiene como función principal poder transmitir un análisis de determinados aspectos a partir de unos datos existentes.

Es posible diferenciar:

- Cabecera
- Cuerpo
- Pie

Para incluir una nueva sección, solo es preciso pulsar la zona del informe mediante el botón derecho o a través de la opción del menú *Report*.

Cada una de las secciones creadas debe tener su propia personalización. Las distintas propiedades pueden ser accesibles tanto a nivel de menú, como a nivel de menú contextuales.

Añadir cabecera/pie

Es posible crear diferentes secciones haciendo uso de la agrupación de datos, llevando a cabo su correspondiente cabecera y pie.

Cada una de las agrupaciones debe tener una cabecera y un pie determinado que se va a utilizar o no.

Para crear una agrupación de datos en el informe es necesario definir el campo o campos sobre los que se desea llevar a cabo la operación. Si la agrupación se va a desarrollar sobre varios campos, es muy importante el orden en el que se va a realizar la aplicación junto con sus implicaciones.

El encabezado y pie de la agrupación se sitúan en la fila superior e inferior adjuntas a los datos.

Por último, cuando se crean grupos, en ocasiones puede llevar a confusión cuando existen distintas páginas, ya que se puede perder visibilidad del campo por el que se agrupa.

5.2. Creación de parámetros

Se utilizan los parámetros para aumentar la funcionalidad de un informe, ofreciéndole una mayor flexibilidad cuando tenga que mostrar los datos. Los parámetros permiten crear informes activos adaptando la información a los valores que se facilitan.

Es posible llevarlos a cabo mediante la ventana de propiedades de datos que se encuentra dentro del propio informe.

5.2.1. Propiedades del parámetro

Hacen referencia a los distintos aspectos, entre otros:

- El tipo de valor que se recibe.
- La descripción asociada a una determinada petición.

5.2.2. Valores de un parámetro

Se realizan estableciendo una respuesta libre del cliente, o bien, ofreciéndole a este una lista de posibles valores para que elija uno. Se pueden obtener las listas a partir de una base de datos o, por otro lado, creando un conjunto cerrado de diferentes elementos que se pueden seleccionar.

5.2.3. Valores predeterminados

Es una opción muy frecuente para garantizar un funcionamiento válido del informe en caso de no disponer de ningún otro valor.

Puede que el programador no haya tenido en cuenta esta opción, por tanto, se podría producir una excepción no controlada.

5.2.4. Actualización de informes

Es conveniente que en un informe en el que se están incorporando diferentes parámetros, se proceda a *refrescar* para que los datos se vayan almacenando. De esta forma, cuando el cliente solicite alguna respuesta de forma inmediata, es fundamental que obtenga los datos más actuales, ya que se habrá actualizado el proceso.

5.3. Creación de subinformes

Ofrece la **posibilidad de crear una relación jerárquica entre varios informes**. Estos informes van a actuar como hijos. El informe principal y los subordinados pueden tener una unión basada en los diferentes datos, de tal forma que el informe principal puede facilitar una serie de datos al subordinado mediante los parámetros correspondientes al subinforme.

Se puede iniciar a partir de dos **dataset** en los que existe un elemento de enlace. A partir de este elemento, aparece la opción de poder relacionar la distinta información que exista entre ellos.

- En un primer momento, cuando el lienzo esté aún sin escribir, se puede definir la estructura del informe. Los elementos más frecuentes van a ser, entre otros, las tablas, los listados y las matrices.
- Llegado el momento de crear un informe, si se selecciona la estructura de una tabla, aportará más velocidad que las demás estructuras disponibles, y se procederán a rellenar los siguientes campos de forma automática simplemente arrastrando y soltando desde el *dataset*.
- A continuación, se debe hacer uso de un informe que ya ha sido creado reforzando su argumento para la reutilización. Mientras se está generando el subinforme, es fundamental centrarse en su aspecto visual. El resto del informe se puede dejar igual, ya que se comporta como un informe más con su origen de datos filtrado, parámetros, formato, etc.
- Después, se definen las posibles vías de comunicación entre el informe principal y el subinforme haciendo uso de la definición de parámetros. Esta definición de parámetros puede afectar al origen de datos, por lo que será conveniente parametrizar la consulta.
- En el informe principal, cuando ya se ha insertado el *subreport*, se puede determinar dónde obtener los valores facilitados como parámetros. De esta forma, se produce la comunicación entre informe principal y subinforme. También será fundamental que el nombre del parámetro definido como entrada en el subinforme, y el nombre del parámetro que se facilita, sean iguales. De no ser así, no se obtendrán los resultados esperados.

5.4. Imágenes

Gracias al uso de las imágenes es posible conseguir un informe más profesional y formal, especialmente si se hace uso de los diferentes logos corporativos o similares. Estas imágenes permiten comprender de mejor forma el informe y hacerlo más atractivo.

5.5. Informes incrustados y no incrustados en la aplicación

Integración de informes

Gracias a los controles incrustados es posible hacer una integración de informes en una determinada aplicación. Para ello, se puede hacer uso de un contenedor dentro de la aplicación que mostrará el contenido del informe. Este contenedor debe ser capaz de interpretar el informe, además de mostrar cómo se ha definido, layouts, parámetros, filtros, etc.

El primer paso en una aplicación *WPF* es utilizar el contenedor que se denomina *WindowsFormHost*. Este contenedor ofrece la posibilidad de hacer uso de controles *Windows Form*.

Será necesario añadir el control *ReportViewer* referenciándolo, es decir, se identificará mediante ensamblado:

Microsoft.ReportViewer.WinForms

Para hacer la selección existen una serie de librerías con el mismo nombre, aunque con versiones diferentes. Es importante elegir la versión adecuada para que, de esta manera, no se obtengan resultados erróneos.

Cuando se realiza la codificación, hay que tener en cuenta una serie de elementos como son la definición de las propiedades de control *reportViewer*, el uso de las propiedades *LocalReport*, en la que es posible seleccionar dónde comienzan los datos en el informe, entre otros.

Otra opción a tener en cuenta es el acceso directo al informe en cuestión. En este caso, *Reporting Services* puede tener acceso mediante el uso de la *http* para visualizar dicho informe. También es posible visualizarlo mediante el uso de controles que actúan como contenedores, para, de esta manera, mostrar diferente contenido web.

5.6. Herramientas gráficas integradas y externas al IDE

Los gráficos son unas herramientas bastante utilizadas, ya que permiten al usuario obtener información importante de una simple pasada. En la mayoría de los casos, los elementos gráficos se utilizan en informes globales que están dirigidos a conocer el estado de situación, mientras que la utilización de las estructuras tabulares y detalladas van dirigidas a actividades de un nivel menos importante.

A la hora de detallar su forma de funcionar dentro de *Reporting Services*, se puede partir de un informe vacío en el que se crea un gráfico que permite conocer el porcentaje correspondiente a las distintas profesiones de un departamento de Recursos Humanos en una determinada empresa.

5.7. Filtrado de datos

Mediante el filtrado de datos es posible especificar los filtros que se necesitan para aquellos datos que se han extraído.

Dependiendo del funcionamiento, lo más conveniente es llevar a cabo, desde el principio de consultas, las que más se adecuen al tipo de información que se pretende mostrar en el informe.

Existen una serie de herramientas centradas en el diseño de informes que permiten perfeccionar los datos que se están utilizando hasta conseguir acercarse a las necesidades especificadas por el diseñador.

Propiedades del Dataset

Partiendo del objeto *Dataset* que se vincula al proyecto, es posible contar con un conjunto de diferentes estructuras y datos que se pueden seleccionar a partir de las bases de datos.

En el caso en el que se realice alguna consulta sobre la ventana de datos de un informe en cuestión, es posible llevar a cabo la identificación de los nombres, además de las distintas características que se puedan emplear en el informe.

Esta estructura ***DataSet*** ofrece la posibilidad de realizar las configuraciones pertinentes según la estructura que se pretenda conseguir, pudiendo filtrar los datos, llevar a cabo diferentes operaciones, etc.

Es posible aplicar los filtros deseados sobre el *DataSet* mediante las expresiones personalizables o mediante la utilización del propio campo.

5.8. Numeración de líneas, recuentos y totales

Existen una gran cantidad de funciones internas del diseñador que pueden ofrecer otras **funciones resumen** de una serie de datos. Estas funciones resumen están asociadas a las operaciones de agrupación de datos que se pueden utilizar, como, por ejemplo, en sentencias SQL cuando se incluye una cláusula *Group By*.

A continuación, se muestra un ejemplo para ver el funcionamiento de la numeración de líneas:

Para crear un informe que señale el número de líneas que tiene cada elemento:

Una vez decidida la información que se desea mostrar, una de las opciones sería:

- Realizar una consulta de datos que incluya el valor a buscar. Esta es una opción poco eficiente ya que se pierde mucho tiempo.
- Por tanto, es posible separar los datos y su origen de la representación de los mismos.

En primer lugar, se puede crear una columna con la organización actual para, a continuación, realizar una búsqueda sobre una función o campo que ofrezca la posibilidad de mostrar la información deseada.

El valor que hay que tener en cuenta es un parámetro que facilita la expresión *RowNumber*, en la que se puede especificar el conjunto de datos para, de esta forma, poder contar las líneas correspondientes.

Seguidamente, se puede incorporar al informe un nuevo valor (al pie del informe), en el que se lleve a cabo la suma de todos los Bonus que se hayan generado.

Otro procedimiento que completaría el anterior: visualizar las diferentes opciones de esta tarea mediante las distintas opciones de herramientas.

5.9. Librerías para la generación de informes. Clases, métodos y atributos

Se dispone de una *API* proporcionada por *Reporting Services*, que cuenta con un gran número de funciones que permiten al desarrollador la integración en distintos tipos de aplicaciones. Es posible utilizarla en los diferentes sitios que se detallan:

- **Servicio web con todas las funciones**

Ofrece una completa funcionalidad del servidor a través de una serie de informes que cuentan con protocolo SOAP (*Simple Object Access Protocol*).

Este servicio web, mediante diferentes métodos, ofrece las diferentes propiedades del servidor de informes y, además, permite crear nuevas herramientas personalizadas para poder utilizar en cualquiera de las partes del informe.

- **Comandos basados en URL**

Se utiliza principalmente para introducir informes en una página web. En este caso, tanto las peticiones como el envío de comandos al servidor de informes, se llevan a cabo mediante la URL.

- **WMI (Windows Management Instrumentation)**

A través de la API que facilita el sistema operativo, es posible hacer uso de una serie de métodos (expuestos por la API) que dependerán de los mecanismos de acceso que estén definidos por la WMI.

- **Extensiones modulares**

Para la arquitectura que se lleva a cabo a la hora de crear, diseñar y visualizar informes por parte de Reporting Services, se hará uso de la personalización de las distintas clases.

- **Programación Report Definition Language**

Es posible declarar un informe mediante RDL (*Report Definition Language*) a través del lenguaje XML.

De esta manera, es posible realizar las modificaciones pertinentes y configuraciones sobre el informe, tanto en su aspecto como en sus funciones.

UF2. Preparación y distribución de aplicaciones

1. Realización de pruebas

La realización de diferentes pruebas es un apartado muy importante cuando se lleva a cabo un proceso de desarrollo. Mediante estas pruebas, es posible verificar que el producto que se ha diseñado no presenta errores y, sobre todo, que desempeña las distintas tareas para las que ha sido diseñado de forma correcta.

1.1. Objetivo, importancia y limitaciones del proceso de prueba. Estrategias

Ámbito de las pruebas. Limitaciones

Una vez que se ha diseñado un producto determinado con un objetivo específico, llega el momento de probar si funciona de forma correcta. Aunque se realicen las pruebas más importantes, será prácticamente imposible llevar a cabo todas las comprobaciones del producto en cuestión.

Por este motivo, se diferencian una serie de conjuntos y particiones que consiguen solventar el mayor número de casos posibles.

Tipos de pruebas

Es posible diferenciar entre dos categorías diferentes:

- **Prueba caja negra:** tienen como objetivo principal especificar las entradas y salidas que produce un determinado producto. En este caso, no es necesario conocer su implementación y se pueden diferenciar los siguientes tipos:
 - Análisis de valores límite.
 - Particiones de equivalencia.
 - Pruebas de comparación.
 - Grafos causa-efecto.
- **Prueba caja blanca:** se tiene en cuenta el valor opuesto a las distintas pruebas que se han realizado, comprobando los elementos internos que son:
 - Cobertura sentencias.
 - Cobertura decisiones.
 - Cobertura condiciones.
 - Cobertura caminos.

1.2. Prueba unitaria

Las pruebas unitarias pretenden comprobar que el código funciona de forma correcta teniendo en cuenta todas las especificaciones que se le han indicado al principio. Para llegar a conseguir esto, se irá comprobando cada módulo de manera individual.

Descripción de la prueba

- Permite hacer particiones en los módulos para que, al ser más pequeños, sea más fácil realizar las pruebas oportunas.
- Cada unidad debe definir sus correspondientes casos de prueba.
- Algunos de los aspectos más importantes para tener en cuenta son: las rutinas de excepción y de error, manejo de parámetros, validaciones, valores válidos y límites, rangos, etc.

Técnicas: consisten en realizar las comparaciones pertinentes de los módulos esperados con respecto al resultado obtenido.

1.3. Pruebas de integración: ascendentes y descendentes

Las pruebas de integración son una serie de pruebas que se le aplican a los diferentes módulos, pero de forma conjunta, ya que, a veces, pueden aparecer errores a la hora de unirlos, o devolver una serie de valores que no son los esperados.

Es posible diferenciar varias técnicas de combinación:

- Si se mezclan todos los módulos y, por tanto, se ejecutan todos de forma conjunta, puede dar lugar a una situación un tanto caótica con numerosos errores.
- Es posible añadir más incorporaciones, pudiendo ser en forma: ascendente o descendente.
- **Ascendente:**
 - Se pueden realizar distintas combinaciones de aquellos módulos de bajo nivel que tengan asignadas tareas determinadas.
 - Es posible diseñar un programa que controle las distintas pruebas y, de esta forma, pueda llevar un control más exhaustivo sobre la entrada y salida de las diferentes pruebas.
 - Permite la comprobación del funcionamiento del grupo.
 - Está capacitado para eliminar los distintos gestores y permite movilidad a los grupos existentes por toda la estructura del programa.

- **Descendente:**

- Permite utilizar el módulo de control principal para llevar un control sobre la prueba en cuestión.
- Según el enfoque que se vaya a seguir, existe la posibilidad de poder sustituir, uno por uno, los resguardos existentes por módulos reales.
- Se realizan diferentes pruebas cada vez que se integren módulos nuevos.

Cada vez que se generen distintos casos de pruebas, por cualquiera de las dos integraciones de las que se disponen (ascendente o descendente), se utilizarán pruebas de cajas negras.

1.4. Pruebas del sistema: configuración y recuperación

El objetivo principal de las pruebas del sistema es chequear el correcto funcionamiento del sistema verificando que los elementos se han integrado correctamente.

Para ello, debe cumplir con una serie de puntos:

1. Debe cumplir todas las funciones que se habían definido previamente.
2. Tanto hardware como software deben funcionar de forma correcta.
3. La documentación del usuario debe ser la adecuada.
4. El rendimiento debe ser el esperado junto con las condiciones límites.
5. Debe asegurar su correcto funcionamiento.

A la hora de ir generando las diferentes pruebas, se utilizarán, al igual que en el caso de pruebas anterior, las pruebas de caja negra, comenzando por una serie de pruebas en el inicio del entorno de desarrollo (*pruebas alfa*), y otras que se realizarán posteriormente en el entorno del cliente (*pruebas beta*).

- **Las pruebas de configuración**

Son las encargadas de chequear el sistema para los distintos entornos de configuración (tanto hardware como software). De esta forma es posible asegurar que los niveles marcados se ajustan a los objetivos de negocio.

- **Las pruebas de recuperación**

Mediante las pruebas de recuperación se pretende dar respuesta al sistema en caso de que exista algún error.

1.5. Pruebas de uso de recursos

Mediante las diferentes pruebas de uso de recursos se pretende averiguar cuáles son los elementos críticos del sistema. En estos casos, los más frecuentes, suelen ser, entre otros, la memoria RAM, CPU o el espacio en el disco.

1.6. Pruebas de seguridad

Mediante las distintas pruebas de seguridad es posible realizar las pertinentes comprobaciones sobre los mecanismos utilizados para, de esta forma, no permitir el acceso a otros mecanismos que, aunque no estén autorizados, funcionen de forma correcta.

1.7. Pruebas funcionales

Tienen como objetivo principal comprobar que el sistema que se ha desarrollado cumple con todos los requisitos que se le habían especificado. Estas pruebas hacen uso de un equipo de pruebas que ha sido creado por los diferentes analistas de pruebas y se lleva a cabo en la etapa final del proceso. En caso de realizarse con éxito, pasaría a la parte de producción.

1.8. Pruebas de simulaciones

Hacen referencia a un conjunto de herramientas que permiten diseñar y ejecutar las diferentes pruebas de carga. Mediante estas pruebas, es posible comprobar cómo va a responder el sistema ante distintas condiciones de trabajo, es decir, permite simular el comportamiento de la aplicación.

Las diferentes pruebas que se pueden encontrar son las que hacen referencia a, entre otras, humo, tensión de carga, rendimiento de la aplicación y diseño de la capacidad.

A la hora de cargar los datos, existen una serie de modelos que pueden llevar a cabo la simulación:

- **Constante:** utiliza siempre la misma carga de usuario en un determinado espacio de tiempo.
- **Paso:** mientras aumentan los valores de carga, supervisan la aplicación.
- **Basado en objetivos:** muy parecida a la anterior, aunque, en este caso, puede seleccionar cuándo se va a detener el aumento de la carga.

1.9. Pruebas de aceptación

Se llevan a cabo cuando el producto ya está preparado para poder implementarse. Estas pruebas las realiza el usuario pudiendo ser ayudado por diferentes personas que forman parte del equipo de prueba.

Las principales características de estas pruebas son:

- Pueden participar de forma activa con el usuario.
- Pueden demostrar si el usuario cumple los requisitos o no.
- Se realizan en la etapa final del proceso de desarrollo.

1.10. Pruebas alfa y beta

La **prueba alfa** la lleva a cabo el desarrollador utilizando un determinado grupo de usuarios finales y anotando cualquier tipo de problema que se presente.

En el caso de la **prueba beta** es el usuario el que no dispone de esta información por lo que se encuentra ante un entorno que no controla. En este caso, es el cliente el encargado de notificar los posibles errores que puedan aparecer para, posteriormente, transmitirlos al equipo.

1.11. Pruebas manuales y automáticas

A la hora de dirigir el desarrollo o control de calidad de un producto software y una vez concluido todo el proceso de implementación, llega el momento de realizar el plan de pruebas. Como se ha visto en apartados anteriores, existen diferentes tipos de pruebas. Será preciso aplicar la mejor manera de llevar a cabo estos tipos de pruebas; si de una manera **automática** (cuando el usuario no interviene en el proceso); o de forma **manual** (donde el usuario es el principal protagonista del plan).

Cada una de ellas presenta una serie de ventajas e inconvenientes.

- Por ejemplo, si se realiza la prueba de forma automática, es posible sustituir las horas que emplearía el usuario, junto con su correspondiente coste, por el tiempo de ejecución del equipo. En caso contrario, se dejaría el control de la prueba a una máquina, con su correspondiente tasa de error.

- Por otra parte, si se realiza la prueba de forma manual, es posible tener un control exhaustivo del plan de prueba, en contraposición con las horas de trabajo de un usuario técnico y experto en la materia, además de los gastos que esto genera.

Sobre este aspecto, existen diferentes vertientes y opiniones sobre la materia. En este caso, este manual se basa en el siguiente cuadro del libro *Agile Testing* que lo recomienda de la siguiente forma:

<http://www.javiergarzas.com/wp-content/uploads/2011/04/agile-test-quad.jpg>

- En el **primer cuadrante** (debajo-izquierda) se recomiendan las pruebas a automatizar, que son las unitarias y de recursos del sistema.
- En el **segundo cuadrante** (arriba-izquierda) se deja a decisión del usuario las pruebas que se pueden realizar de tipo manual y automático, como pueden ser las pruebas funcionales, de capacidad y rendimiento.
- En el **tercer cuadrante** (arriba-derecha) se recomiendan las pruebas manuales: usabilidad y alfa-beta.
- En el **cuarto cuadrante** (abajo-derecha) están las diferentes pruebas de verificación que se llevan a cabo mediante una serie de herramientas destinadas para tal fin.

1.12. Herramientas de software para la realización de pruebas

Pruebas unitarias Visual Studio

Como se ha indicado en apartados anteriores, las pruebas unitarias se encargan de comprobar que un determinado código funcione de forma correcta teniendo en cuenta las especificaciones oportunas.



En el caso de pruebas unitarias sobre el entorno **visual Studio** se siguen los siguientes pasos:

1. Se comienza creando un determinado **Proyecto** partiendo de las distintas plantillas sobre Test disponibles.
2. **Crear un archivo de pruebas:** cuando se crea un archivo de prueba, se automatizan las diferentes opciones de testing para aportar un mayor beneficio y ahorrar todo el tiempo posible. Los diferentes archivos de pruebas pueden tener extensión: *.runsettings*, *.testsettings*, *.testrunconfig*, *.vsmdi*.
3. **Estructura archivo de pruebas:** en este caso se puede seleccionar un tipo de lenguaje para su representación, como puede ser, XML.

2. Documentación de aplicaciones

La documentación de las distintas aplicaciones es una tarea muy importante en cualquier proyecto. Es preciso señalar que, si se cuenta con la documentación adecuada y detallada, es posible acceder a parte del código necesario en un determinado momento para después ser reutilizado en otro proyecto distinto.

Esta tarea de documentación por parte de los desarrolladores es bastante tediosa, por lo que no se le dedica demasiado tiempo. Este es uno de los motivos por el que se cree conveniente automatizar el desarrollo de la distinta documentación de ayuda.

2.1. Archivos de ayuda. Formatos

Ayuda contextual

Ofrece al usuario, a través de diferentes elementos visuales o de texto, un acceso más rápido y directo a la función que desee realizar.

- **ToolTips:** son una serie de elementos que aparecen en la interfaz cuando el ratón se posiciona sobre algún elemento determinado. El objetivo de estos elementos es brindar al usuario alguna ayuda sobre ese elemento en cuestión.
- **Ficheros ayuda:** son aquellos que permiten ofrecer una ayuda al usuario al pulsar la tecla F1. Gracias a estos ficheros de ayuda, se pretende facilitar la tarea del usuario sobre una función específica.

Uno de los comandos más utilizados es:

- **ApplicationCommands.Help:** es una clase a la que pertenecen los diferentes comandos más frecuentes, como pueden ser, entre otros, las operaciones de apertura y cierre de ventanas, imprimir, copiar y pegar.

2.2. Herramientas de generación de ayudas

A la hora de desarrollar un código, es posible ayudarse de comentarios que se detallan sobre este mismo código para ofrecer documentación sobre los temas tratados. Se conoce como meta-información y se escribe precedido de los caracteres `///` delante del código.

Por ejemplo, es posible detallar una serie de comentarios sobre un determinado código y se escribe de la siguiente manera:

```
///
    ...

```

Herramientas GhostDoc

GhostDoc corresponde a la extensión de *Visual Studio* que permite generar, de forma automática, documentación XML que contiene toda la información sobre los comentarios de un código determinado, junto con sus principales funciones.

Esta herramienta hace uso de la documentación correspondiente que le facilita Microsoft o el proveedor de la librería utilizada.

- **Instalación Ghostdoc:** a la hora de crear la documentación en XML que se encuentra dentro de las propiedades de un proyecto determinado, se siguen los siguientes pasos:
 - En primer lugar, se accede a la opción compilar (Build).
 - A continuación, se selecciona la opción de archivo de documentación XML.
 - Y, por último, se define el nombre junto con la ruta correspondiente.

Herramientas Sand Castle

En este caso, la herramienta SandCastle la crea *Microsoft* y se utiliza también para crear documentación correspondiente a un documento determinado, pero, basándose en *estilo MSD de .NET* con los comentarios XML que estén asociados.

La utilización de esta herramienta es bastante sencilla y está basada en la línea de comandos, por lo que no cuenta con una GUI definida de antemano.

- **Instalación Sandcastle:** a la hora de instalar la herramienta *SandCastle*, se necesita, previamente, instalar una serie de paquetes que son requisito indispensable. Por este motivo, esta herramienta cuenta con un asistente de aplicación que hace más fácil este proceso.
- **Interfaz gráfica Help File Builder:** la interfaz gráfica permite que el usuario pueda tener todo más accesible mejorando el entorno visual, por lo que no

es necesario que tenga conocimientos específicos para trabajar con ella. Llegado el momento de introducir los comentarios en la documentación, puede que se necesite recurrir a la instalación de otros programas que permitan generar estos archivos.

Herramientas HelpNDoc

Es una herramienta alternativa a la anterior que permite generar diferentes archivos de ayuda en formatos como: *chm*, *pdf*, *html*, entre otros.

Además, cuenta con algunas versiones gratuitas que se pueden descargar desde el siguiente enlace:

<http://www.helpndoc.com/download>

Una vez que está descargada e instalada, los pasos a seguir son:

1. Se crea un nuevo proyecto.
2. A continuación, se seleccionan los datos correspondientes al nombre, idioma y tabla de contenidos.
3. Se realizan las configuraciones pertinentes de esta herramienta.

2.3. Tablas de contenidos, índices, sistemas de búsquedas, entre otros

Con las tablas de contenidos se puede elaborar la estructura correspondiente de un determinado proyecto, y en las tablas se pueden identificar los títulos de los distintos temas o subtemas que intervienen en el proyecto. Asimismo, existe la opción de que la tabla de contenidos contenga el número de página o no.

Una vez elaborado el índice mediante las tablas de contenidos, es posible tener acceso a un lugar concreto, indicando el número de página en la que se encuentra.

2.4. Tipos de manuales

Durante el tiempo en el que se desarrolla un proyecto, se van generando diferentes tipos de documentación que deben entregarse a los usuarios correspondientes para que cuenten con la información determinada del producto desarrollado.

A continuación, se definen algunos de estos manuales:

Manual de instalación

En él se determinan una serie de detalles necesarios a la hora de realizar la instalación como pueden ser, entre otros, los diferentes aspectos que debe tener el software según el sistema operativo que se utilice, las aplicaciones que es necesario tener instaladas, las características que debe tener la parte hardware, la cantidad de memoria RAM disponible, etc.

Se puede desarrollar la documentación en dos versiones diferentes:

- **Reducida:** se lleva a cabo una instalación rápida con los valores seleccionados por defecto sin entrar en muchos más detalles.
- **Detallada:** esta versión explica de forma más detallada todos los aspectos que intervienen en el proceso de instalación.

Documentación de configuración

En el caso de la documentación de configuración se deben detallar todos aquellos parámetros que tengan que ver con la configuración de una determinada aplicación, además de todos aquellos cambios que se puedan producir en el momento en el que funcione la aplicación.

En este tipo de documentación se tratan distintos puntos de vista como pueden ser los aspectos software, hardware, aspectos visuales, etc. Además, se define la forma en la que se van a ejecutar las diferentes aplicaciones o librerías que va a necesitar el proyecto en cuestión. Asimismo, se tienen en cuenta aquellos aspectos que pueden variar y, por tanto, las consecuencias que se van a producir en el funcionamiento de la aplicación.

Cuando se habla de la documentación de configuración es bastante frecuente diferenciar entre dos niveles distintos:

- Configuración **normal**.
- Configuración **avanzada:** es muy parecida a la anterior, aunque, en este caso, se detallan de forma más exhaustiva distintos aspectos más complejos que necesitan que los usuarios tengan una mayor experiencia y conocimiento (administradores).

Manual de usuario

Al igual que en otros modelos de manuales, en el manual de usuario también se diferencian dos versiones:

- **Rápida.**
- **Detallada:** esta versión amplía las diferentes opciones a la hora de realizar las diversas posibilidades de aplicación.

En algunos casos, puede que esta información la lleve a cabo la interfaz de usuario de una aplicación determinada. De esta forma, es preciso indicar al usuario los diferentes pasos que se deben seguir para poder desarrollar un trabajo mediante las ventanas/pantallas/páginas de la interfaz de usuario. Así, el usuario final debe poder identificar las imágenes en la interfaz real de la aplicación en cuestión.

Según el nivel de los usuarios del sistema existen distintos tipos de versiones: principiante, inicial, medio, avanzado, experto. Según el caso, se profundizará más o menos en las diferentes aplicaciones.

Guía de usuario

El objetivo principal de la guía de usuario es el mismo que el anterior (manual de usuario), con la diferencia de que, en este caso, cambia la extensión y la cantidad de información disponible.

Guía rápida

En esta guía rápida se dispone de un resumen con todos los aspectos más importantes que se han desarrollado en la documentación previa.

Manual de administración

Este manual detalla la instalación junto con la administración del sistema. Además, en él viene detallado el proceso de configuración de aquellos componentes que forman parte del sistema.

Las personas encargadas de desarrollar estos manuales suelen ser los administradores, ya que son los que tienen otorgados más permisos sobre el proceso.

3. Distribución de aplicaciones

Una de las funciones más importantes que se encuentran integradas en las herramientas *IDE* es el despliegue de paquetes.

Entre sus funciones principales es posible destacar la encargada de generar un archivo ejecutable tras realizar el proceso de compilación, creando diferentes paquetes de instalación mediante un asistente guiado.

Gestión de versiones

La mayoría de los productos software van evolucionando con el paso del tiempo, mejorando siempre los posibles inconvenientes que se hayan encontrado. Para ello, se van generando nuevas versiones o añadiendo nuevas funcionalidades a versiones ya existentes.

Existe un sistema de versionado que depende de una serie de factores, como pueden ser, entre otros, el tipo de producto y el fabricante. En la mayoría de los casos, suele incluir, además, la secuencia ***major.minor***.

Para la numeración de las diferentes versiones se suele utilizar la definición de las versiones impares como versiones beta, y las pares como versiones liberadas.

Según el número de versión que corresponda a un determinado producto, los distintos informes se llevarán a cabo siguiendo un sistema de identificación de versiones:

<i>Major</i>	<i>Minor</i>	<i>Build</i>	<i>Revisión</i>
--------------	--------------	--------------	-----------------

3.1. Componentes de una aplicación. Empaquetado

Paquetes

Existen distintos paquetes que constan de componentes individuales, mientras que otros están formados por un conjunto de aplicaciones relacionadas entre ellas. Una vez que se crea un paquete ya es posible distribuirlo a otros usuarios y organizaciones. Clasificación de paquetes:

- **Paquetes gestionados**

Los paquetes gestionados se utilizan principalmente para vender o distribuir diferentes aplicaciones a los clientes. Es conveniente que estos paquetes se creen a partir de una organización *Developer Edition*. De esta forma, los desarrolladores encargados de crear estos paquetes tienen control para realizar la gestión y venta de los mismos.

No tienen ningún tipo de problema en las distintas actualizaciones, por lo que, cuando aparecen algunas modificaciones o eliminaciones que pueden ser destructivas, no las realizan para no correr riesgos. Además, cuentan con una serie de **ventajas**, entre las que destacan:

- Protegen la propiedad.
- Son compatibles con otras versiones.
- Cuentan con capacidad para añadir divisiones o parches a versiones anteriores.
- Son capaces de enviar actualizaciones de parches y permiten asignar nombres únicos a cada uno de sus componentes para, de esta forma, evitar conflictos en el proceso de instalación.

- **Paquetes no gestionados**

Aquellos paquetes que no están gestionados suelen utilizarse para una serie de proyectos de código abierto, por lo que pueden brindar a los distintos desarrolladores los fundamentos principales de una determinada aplicación.

Cuando se han instalado los componentes relativos a un paquete sin gestionar, es posible realizar las modificaciones oportunas en la organización en la que se han instalado.

El desarrollador encargado de crear el paquete sin gestionar no es el que tiene asignado el control de aquellos componentes instalados, por tanto, no puede modificarlos.

Componentes

Los componentes son el conjunto de elementos que forman un paquete, teniendo en cuenta que un elemento es un objeto personalizado. Existe la posibilidad de unificar distintos componentes de un paquete para conseguir crear determinadas funciones.

En el caso de los paquetes sin gestionar, los componentes que lo forman no se pueden actualizar. Sin embargo, en los paquetes gestionados, es posible actualizar solo algunos de sus componentes, pero no todos.

Firma de componentes

Existen una serie de componentes de los que debe verificarse su origen para poder garantizar su autenticidad. Esta verificación se puede realizar a través de la utilización de la firma.

3.2. Instaladores

Creación de paquetes de instalación

Cuando se va a publicar un paquete de instalación, es posible especificar el lugar desde el que se va a realizar, como puede ser, por ejemplo, un servidor web o FTP.

Después de realizar la instalación en un paquete, es posible que haya que implementar una serie de componentes del paquete en cuestión para que esté disponible para los distintos usuarios que forman parte de la generación.

El modo de instalación de un proyecto es el que define si es accesible desde un servidor web, desde solo un punto de la red, o desde el propio escritorio mediante la utilización de diferentes menús.

Por otra parte, los paquetes de actualización también incorporan configuraciones importantes que permiten crear soluciones personalizadas. El paquete de instalación dispone de un repositorio donde se publican las diferentes versiones de los archivos de instalación junto con sus correspondientes instalaciones, y al que el cliente puede acceder mediante un enlace determinado.

Otro de los aspectos para tener en cuenta en la creación de paquetes de instalación es el listado de los archivos a instalar, así como los pre-requisitos que se deben cumplir antes de comenzar con este proceso (como puede ser la instalación **ds**

frameworks). Estas condiciones se deben seleccionar a partir de una lista desplegable en la que aparezcan los siguientes pre-requisitos:

- **Opciones de publicación de software**

Las diferentes opciones de publicación se dividen en cuatro secciones. Cada una está dedicada a un aspecto concreto del despliegue.

- **Descripción**

En esta sección es posible definir los datos básicos del archivo de publicación, como puede ser el nombre, idioma de publicación, configuración y carpeta que contendrá el software dentro del menú de inicio.

- **Implementación**

En este apartado se puede señalar un dato clave, como puede ser la creación de la página web, a partir de la cual se realiza este despliegue, así como la extensión del paquete o verificaciones de integridad de los distintos archivos, que puede ser, entre otras, las opciones de comenzar de forma automática al insertar un CD.

- **Manifiesto**

En este apartado se define el comportamiento de la aplicación a la hora de controlar el acceso. Una de las opciones es limitar la ejecución a través del menú de inicio, o bien permitir dicha ejecución a través de una URL de distribución. Ofrece también la posibilidad de crear un acceso directo en el escritorio, desde el que poder ejecutar dicho programa.

- **Asociaciones de archivo**

En este apartado es posible realizar las configuraciones del icono de la aplicación, así como la extensión del sistema.

- **Asistente de instalación y desinstalación**

Llegados a este punto, ya es posible generar el archivo **setup** en la opción generar de *Visual Studio*.

En un paso previo a este, se ha tenido que configurar en el editor la ruta donde se van a almacenar todos los archivos **setup**. En caso de no realizarse, se almacenarán en una ruta por defecto dentro de la carpeta del programa *Demo*.

Una vez generado el archivo es preciso ir a la carpeta repositorio, en la que se encuentran todos los archivos instalables, para seleccionar aquellos archivos deseados.

Estos archivos ya son los instalables que se pueden facilitar a aquellas personas que se interesen por el programa en cuestión. Para este manual y, como simulación, se procede a la instalación accediendo al archivo ejecutable o aprovechando que el editor se encuentra abierto, en la opción **Proyecto/instalar**, en la que también se puede ejecutar el instalador.

Una vez realizado este paso de alguna de las dos maneras detalladas anteriormente, se inicia el asistente de instalación con las opciones que se han definido a la hora de crear el paquete instalador.

En primer lugar, aparece una ventana con la información y advertencias de los requisitos que hay que cumplir.

Tras pulsar en **Siguiente**, aparece en el asistente la ventana gráfica en la que es posible elegir el directorio o carpeta de instalación, donde se puede seleccionar la carpeta que se ha diseñado por defecto o se procede a la creación y elección de una nueva carpeta.

Nuevamente se hace clic en **Siguiente** y aparece la ventana con una lista de los requisitos previos que se deben tener instalados antes de llevar a cabo este proceso y/o la opción de instalar estas funciones previas. Estos requisitos, como se ha dicho anteriormente, ya se han seleccionado a la hora de crear el paquete instalador.

Se vuelve a seleccionar **Siguiente** y aparece la confirmación de los pasos anteriores y su posterior instalación de la aplicación.

Una vez completada la instalación, dependiendo de las opciones con la que se haya creado el paquete instalador, se creará el acceso directo en el escritorio seguido del icono seleccionado.

3.3. Paquetes autoinstalables

En esta opción aparece la herramienta **NuGet** que, al ser de código abierto, simplifica bastante la tarea del empaquetado de paquetes junto con su distribución. En este caso, la persona encargada de desarrollar el código del programa no tiene por qué ser el que descargue, descomprima y lleve a cabo las diferentes opciones referentes al proyecto.

Las principales tareas que desarrolla la herramienta **NuGet** son las detalladas a continuación:

- Descargar el archivo que contiene el paquete.
- Extraer el contenido del paquete.
- Realizar las referencias correspondientes a los ensamblados.
- Copiar el contenido completo del proyecto.
- Aplicar una serie de características específicas de cada paquete.
- Ejecutar los distintos scripts de automatización cuando sea necesario.

Creación de paquete con NuGet

Crear paquetes a través de la herramienta NuGet es un proceso bastante sencillo ya que solo hay un ensamblado para representar un componente.

A continuación, se establecen los pasos a seguir para este proceso:

1. Se crea un proyecto tipo *class library*.
2. Seguidamente, se genera el manifiesto NuSpec para el proyecto: *NuSpec* es el archivo correspondiente al paquete que tiene el manifiesto de los metadatos básicos (autor, descripción, dependencias, etcétera). Debe tener un formato simple para que todos puedan acceder a él y se ejecuta desde el mismo directorio en el que se encuentra el proyecto.

Nuget spec

3. Ya es posible actualizar los metadatos del ensamblado del proyecto: el fin de actualizar estos datos es asegurar que no van a existir discrepancias entre este y el que se genera en el punto anterior.
4. Haciendo uso del comando *NuGet.exe*, se crea el paquete: mediante el comando *nugetpack*, que se encuentra en el mismo directorio del proyecto y del archivo *NuSpec*.

Nugetpack nombreProyecto.csproj

Creación paquete NuGet utilizando interfaz gráfica

Otra forma que existe a la hora de crear un paquete NuGet es haciendo uso de una interfaz gráfica, que es posible obtener de la siguiente dirección:

<https://www.nuget.org/>

Mientras se realiza el proceso de instalación, es posible comprobar los distintos archivos que se encuentran en la dirección *url*, de tal forma que los clientes solo descargan la instalación y no es necesario que lancen el paquete completo.

Una vez instalado, ya es posible crear paquetes de forma sencilla mediante la utilización de tareas comunes o a través del menú archivo.

Los datos pertenecientes a un paquete concreto que se pueden editar (*metadatos*), permiten que el desarrollador haga uso de sus valores a través de la interfaz.

3.4. Herramientas para crear paquetes de instalación

Existen una serie de programas, como por ejemplo Visual Studio, que anteriormente disponía de unas plantillas que se podían utilizar para generar los paquetes de instalación. Sin embargo, a medida que han ido avanzando con nuevas versiones, se han eliminado estas utilidades del editor.

Por tanto, es necesario buscar una solución alternativa para la creación de paquetes de instalación. Una de las herramientas más populares es la aplicación WiX (*Windows Installer XML*).

Esta herramienta permite crear paquetes de instalación de Windows utilizando los conceptos vistos anteriormente y, una vez que está compilado, es posible generar su ejecutable correspondiente.

Para descargar y consultar más información, es posible acceder a la página:

<http://wixtoolset.org>

En esta herramienta se pueden crear distintos paquetes a partir de código *XML*. Se encuentra bien conectado con el editor *Visual Studio* y, una vez que está instalada, es posible utilizarlo para crear WiX como una extensión más. De esta forma, se

pueden utilizar las diferentes plantillas que van a aparecer en las opciones a la hora de crear un nuevo proyecto.

A modo de ejemplo, se puede abrir un archivo por defecto que venga cargado en el editor para que, de esta forma, se pueda comprobar el esquema básico de estos tipos de archivos.

```
<Wix>           //Raíz del archivo

    <Product>     //Creación de un producto que representa al instalador

<Package>       //Comprensión del paquete

<Media>         //Dispositivo en el que se divide la instalación. Casi
                siempre es un solo dispositivo.

<Directory>     //Ruta en la que se va a instalar el archivo

<Component>    //Componente de la instalación

<File>          //Ficheros que se van a instalar

<Feature>       //Configuraciones disponibles en la instalación: típicas,
                personalizadas, detalladas
```

A lo largo de la instalación, se deben poder modificar aspectos del sistema como pueden ser: modificación en el sistema de archivos (con creaciones de carpetas, archivos y acceso directo), modificaciones en el sistema de registro del sistema operativo (creando nuevos registros o modificando los registros ya creados), cambios en los tipos de archivo (se pueden crear asociaciones de archivos en el sistema), así como acciones personalizadas o verificaciones del despliegue (asegurarse antes de la instalación que se cumplen todas las precondiciones).

Todos estos cambios, como es lógico, deben estar representados en el archivo XML que se debe crear.

Para finalizar y, como último paso, es preciso indicar el tipo de archivo de salida del instalador una vez compilado el fichero XML.

Los más frecuentes son: *extensión msi* (Windows installer package) o el archivo *ejecutable.exe* (Executable package).

Bibliografía

Juan Luis Vicente Carro, “Desarrollo de Interfaces”. Garceta, 2014.

Webgrafía

<http://periodico.laciudadaccesible.com>

<http://www.fdi.ucm.es> Universidad Complutense de Madrid. Interfaz Gráfica

```
function updatePhotoDescription() {  
    if (descriptions.length > (page * 9) + (currentimage substituting() - 1)) {  
        document.getElementById("bigimageDesc").innerHTML = descriptions[page * 9 + (currentimage substituting() - 1)];  
    }  
}  
  
function updateAllImages() {  
    var i = 1;  
    while (i < 10) {  
        var elementId = "foto" + i;  
        var elementIdBig = "bigimage" + i;  
        if (page * 9 + i - 1 < photos.length) {  
            document.getElementById(elementId).src = "images/" + photos[page * 9 + i - 1];  
            document.getElementById(elementIdBig).src = "images/" + photos[page * 9 + i - 1];  
        } else {  
            document.getElementById(elementId).src = "images/placeholder.jpg";  
            document.getElementById(elementIdBig).src = "images/placeholder.jpg";  
        }  
        i++;  
    }  
}
```