UD2. Xeración de interfaces gráficas de usuario empregando a linguaxe XML

RA2.

Generar interfaces gráficas de usuario basados en XML utilizando herramientas específicas y adaptando el documento XML creado.

- CA2.1. Se reconocen las ventajas de generar interfaces gráficas de usuario a partir de su descripción en XML..
- CA2.2. Se genera la descripción de la interface en XML usando un editor gráfico.
- CA2.3. Se analiza el documento XML generado.
- CA2.4. Se modifica el documento XML.
- CA2.5. Se les asignaron acciones a los eventos.
- CA2.6. Se generó el código correspondiente a la interfaz a partir del documento XML.
- CA2.7. Se programó una aplicación sencilla para comprobar la funcionalidad de la interfaz generada.

Xeración de interfaces gráficas de usuario empregando a linguaxe XML

- Lenguajes de descripción de interfaces basadas en XML: ámbito de aplicación.
- Elementos, etiquetas, atributos y valores.
- Herramentas libres e propietarias para la creación de interfaces de usuario multiplataforma.
- Controles: propiedades.
- Eventos: controladores.
- Edición y depuración del documento XML.
- Generación de código para diferentes plataformas.

➔ Linguaxes de descrición de interfaces baseadas en XML: ámbito de aplicación.

Un **lenguaje de marcado** o **lenguaje de marcas** es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

A veces se confunde lenguaje de marcas con lenguaje de programación y es un error ya que el primero no contiene ni variables, ni expresiones aritméticas ni sentencias del tipo condicional o iterativo todo típico de un lenguaje de programación al uso.

Entre los lenguajes de marcas o de descripción más importantes y de gran influencia en el campo que nos ocupa, la generación de interfaces gráficas se encuentra el **XML[1].**

XML no ha nacido sólo para su aplicación en **Internet**, sino que se propone como un **estándar para el intercambio de información estructurada** entre diferentes plataformas. Se puede usar y se usa en bases de datos, editores de texto, hojas de cálculo e intefaces gráficas, entre otras...

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Edit_Mensaje SYSTEM "Edit_Mensaje.dtd">

<Edit_Mensaje>
     <Mensaje>
          <Remitente>
               <Nombre>Nombre del remitente</Nombre>
               <Mail> Correo del remitente </Mail>
          </Remitente>
          <Destinatario>
               <Nombre>Nombre del destinatario</Nombre>
               <Mail>Correo del destinatario</Mail>
          </Destinatario>
          <Texto>
               <Asunto>
                    Este es mi documento con una estructura muy sencilla
                    no contiene atributos ni entidades...
               </Asunto>
               <Parrafo>
                    Este es mi documento con una estructura muy sencilla
                    no contiene atributos ni entidades...
               </Parrafo>
          </Texto>
     </Mensaje>
</Edit_Mensaje>
```

Ejemplo de Documento XML

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- Este es el DTD de Edit_Mensaje -->

<!ELEMENT Mensaje (Remitente, Destinatario, Texto)*>
<!ELEMENT Remitente (Nombre, Mail)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Mail (#PCDATA)>
<!ELEMENT Destinatario (Nombre, Mail)>
<!ELEMENT Nombre (#PCDATA)>
<!ELEMENT Mail (#PCDATA)>
<!ELEMENT Texto (Asunto, Parrafo)>
<!ELEMENT Asunto (#PCDATA)>
<!ELEMENT Parrafo (#PCDATA)>
```

Documento DTD

1 Ya que en 1º de DAM existe un módulo dedicado a lenguaje de marcas no entraremos en detalle de sus características.

Entre los lenguajes de descripción de interfaces basados en XML más importantes (no están todos ya que algunos aún se están desarrollando) tenemos:

- **UIML** *(User Interface Markup Language):* es un sencillo lenguaje basado en XML que permite realizar una descripción declarativa de la interfaz de usuario de un modo **independiente del dispositivo**. Para describir una interfaz de usuario en UIML se debe realizar, por un lado la **definición de la interfaz genérica**, y por otro un documento UIML que **representa el estilo de presentación apropiado para el dispositivo** en el cuál la interfaz de usuario se va a ejecutar. De este modo, una misma aplicación solamente necesitará un único documento UIML de especificación válido para cualquier dispositivo y un documento de estilo propio para cada dispositivo.

- **XIML** *(eXtensible Interface Markup Language)* es un lenguaje de especificación basado en XML. Se propone como lenguaje de especificación común e infraestructura de desarrollo para profesionales de la interfaz de usuario en todos los ámbitos, diseñadores, ingenieros de software o expertos en usabilidad.

- **MXML** *(Macromedia)* es un lenguaje descriptivo desarrollado inicialmente por Macromedia hasta el 2005 para la plataforma FLEX de Adobe. MXML se basa en XML y su acrónimo "Macromedia eXtensible Markup Language". Es un lenguaje que describe interfaces de usuario, crea modelos de datos y tiene acceso a los recursos del servidor, del tipo RIA (Rich Internet Application). MXML tiene una mayor estructura en base a etiquetas, similar a HTML, pero con una sintaxis menos ambigua, proporciona una gran variedad e inclusive permite extender etiquetas y crear sus propios componentes. Una vez compilado genera **ficheros .swf.**

- **GladeXML-GtkBuilder** más que un lenguaje es una herramienta de desarrollo visual de interfaces gráficas mediante GTK/GNOME. Es independiente del lenguaje de programación y no genera código fuente sino un archivo **XML.** El **IDE Anjuta** lo lleva integrado para el desarrollo de interfaces pero puede trabajar independientemente. Actualmente está en la versión Glade3 que fue reescrita totalmente aumentando el número de widgets y haciéndolo más ligero. **GtkBuilder** es un formato XML que Glade usa para almacenar los elementos de las interfaces diseñadas. Estos archivos pueden emplearse para construirlas en tiempo de ejecución mediante el objeto GtkBuilder de GTK+. **GladeXML** era el formato que se usaba en conjunto con la biblioteca *libglade* (ambos obsoletos en favor de GtkBuilder).

- **XAML** (*e**X**tensible **A**pplication **M**arkup**L**anguage*) es el lenguaje de formato para la interfaz de usuario para la **Base de Presentación de Windows** y **Silverlight,** el cual es uno de los

"pilares" de la interfaz de programación de aplicaciones .NET en su versión 3.0 (conocida con anterioridad con el nombre clave WinFX).

XAML es un lenguaje declarativo basado en XML, optimizado para describir gráficamente interfaces de usuarios visuales ricas desde el punto de vista gráfico, tales como las creadas por medio de Adobe Flash. En su uso típico, los archivos tipo XAML serían producidos por una herramienta de diseño visual, como Microsft Visual Studio. El XML resultante es interpretado en forma instantánea por un sub-sistema de despliegue de de los sistemas Windows a partir del Vista que reemplaza al GDI de las versiones anteriores de Windows. Los elementos de XAML se interconectan con objetos del Entorno Común de Ejecución. Los atributos se conectan con propiedades o eventos de esos objetos.

XAML fue diseñado para soportar las clases y métodos de la plataforma .NET que tienen relación con la interacción con el usuario, en especial el despliegue en pantalla.

Un archivo XAML puede ser compilado para obtener un archivo binario XAML **.baml,** el cual puede ser insertado como un recurso en un ensamblado de Framework .NET. En el momento de ejecución, el motor del Framework extrae el archivo .baml de los recursos del ensamblado, se analiza sintácticamente, y crea el correspondiente árbol visual WPF o Workflow. Cuando se use en **Windows Presentation Foundation, XAML es usado para describir interfaces visuales para usuarios.** WPF permite la definición de objetos en 2D

y 3D, rotaciones, animaciones y otra variedad de características y efectos. Cuando es usado en el contexto de **Windows Workflow Foundation, XAML es usado para describir lógica declarativa**, como aquellos creados en el proceso de sistemas de modelado y herramientas. El formato de serialización para WorkFlows había sido llamado previamente XOML, para diferenciarlo de su uso en IU de los XAML, pero esa diferenciación ya no existe. Sin embargo las extensiones de los archivos que contienen marcado de workflow es todavía XOML.

Este ejemplo en XAML muestra un texto "Hola Mundo!" dentro de un contenedor del tipo Canvas.

```
<Canvas xmlns="http://schemas.microsoft.com/client/2007"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <TextBlock>Hola Mundo!</TextBlock>
</Canvas>
```

➔ **Xforms** es un formato XMLdiseñado por el W3C para poder definir interfaces de usuario, **principalmente formularios web**. XForms ha sido diseñado para ser la nueva generación de formularios HTML/XHTML, pero es lo suficientemente genérico como para que pueda ser usado, de una manera independiente, para describir cualquier interfaz de usuario e incluso para realizar tareas simples y comunes de manipulación de datos.

Desde el de 2009 y hasta hoy en día la recomendación oficial del W3C es XForms 1.1. Existen varios plugins y extensiones que le dan soporte a diferentes navegadores. Firefox soporta XForms a través de una extensión. Para IE6 está formsPlayer, un plugin que extiende al navegador haciendo que soporte XForms, DOM 2 Events, DOM 3 XPath, XML Events y el DOM 3 Implementation Registry. La versión 2.0 y superiores de Openoffice.org soportan Xforms.

XForms también puede ser usado a través de varias tecnologías de servidor que convierten el código de XForms a formularios de HTML en tiempo de ejecución y de manera transparente. Recientemente ha sido dada a conocer una nueva herramienta, llamada AJAXForms, que transforma, en tiempo de compilación, documentos XHTML/XForms en páginas HTMLcon Javascript, que sí entienden los navegadores actuales. Estas páginas gestionan, sin interactuar con el servidor, tanto la presentación como la lógica de la interfaz de usuario y su comunicación con el servidor se restringe al intercambio de datos utilizando técnicas AJAX.

XSLTForms transforma documentos XHTML/XForms en páginas XHTML con JavaScript en los navegadores con XSLT.

➔ **XUL** (XML-*based* ***U****ser-interface* ***L****anguage*, es la aplicación de XML a la descripción de la interfaz de usuario en el navegador Mozilla. No es un estándar. Destaca por su portabilidad y su independencia de dispositivo. Provee un gran conjunto herramientas para crear menús, paneles, barras de herramientas, wizards, entre otras. Gracias a esto, no será necesario utilizar un lenguaje de programación propietario o incluir un gran código JavaScript para manejar el comportamiento de la interfaz de usuario.

Una interfaz XUL es definida mediante la especificación de tres grupos de componentes distintos:

- **Content:** Aquí se encuentran los documentos XUL, que definen el diseño de la interfaz.
- **Skin:** Contiene las hojas de estilos (CSS) y las imágenes, las cuales definen la apariencia de la interfaz.
- **Locale:** Los documentos DTD se encuentran aquí, estos documentos facilitan la localización de páginas XUL.

Muchas de las interfaces desarrolladas para las extensiones de Mozilla Firefox son basadas en XUL.

➔ **XBL** (*e**X**tensible **B**indings **L**anguage)* es un lenguaje de marcas que se emplea para definir el comportamiento y la apariencia de aplicaciones XUL y elementos XML. El lenguaje XUL define la disposición de la interfaza de usuario de una aplicación, que puede adoptar diferentes aspectos dependiendo del estilo definido. Sin embargo resulta imposible definir cómo funciona cada elemento, como por ejemplo, la forma en que funcionan una barra de progreso. Es aquí donde entra en juego el lenguaje XBL.

Un archivo XBL contiene asociaciones **(*bindings*).** El elemento raíz de todo documento XBL es **<bindings>**, que contiene a su vez uno o varios elementos **<binding>**. Cada uno de estos últimos declara un *binding* que puede asignarse a cualquier elemento XUL. La forma de realizar esta asignación es a través de las hojas de estilo: la propiedad **-moz-binding** del elemento XUL debe indicar la URL del documento XBL.

```
scrollbar {
    -moz-binding: url('somefile.xml#binding1');
}
```

➔ **Elementos, etiquetas, atributos e valores.**

Existen tres términos comúnmente usados para describir las partes de un documento XML: **etiquetas, elementos y atributos.** Aquí está un documento de ejemplo que ilustra estos términos:

*<direccion>**<datos>***

<titulo>Mrs.</titulo>

<nombre> Mary </nombre>

<apellidos> McGoon </apellidos> ***</datos>***

<calle> 1401 Main Street </calle>

<ciudad estado="NC">Anytown</ciudad>

<codigo-postal> 34829 </codigo-postal>

</direccion>

Una **etiqueta** es un texto entre el símbolo menor que (<) y el símbolo mayor que (>). Existen etiquetas de **inicio** (como <nombre>) y etiquetas de **fin** (como </nombre>).

Un **elemento** consta de la etiqueta de inicio, la etiqueta de fin y de todo aquello que este entre ambas. En el ejemplo anterior, el elemento <datos> contiene tres elementos hijos: <titulo>, <nombre>, y <apellidos>.

Un **atributo** es un par **nombre-valor** dentro de la etiqueta de inicio de un elemento. En este ejemplo, estado es un atributo del elemento <ciudad>, en ejemplos anteriores <estado> era un elemento.

```
<html xmlns=" ht tp : / /www.w3 . org /2002/06/ xhtml2 " xml : lang=" sp ">
<head> <title>Busqueda</ title>
<model>
<submission action="http://www.ejemplo.org/busca" method="get " id="busca"/></model>
</head>
<body>
<p></p>
<input ref=" cad "><label>Cadena : </label></input>
<submit submission=" busca "><label>Buscar</label></submit>
</body>
</html>
```

Ejemplo de Xforms (Formulario basado en XML)

➔ Ferramentas libres e propietarias para a creación de interfaces de usuario multiplataforma.

Las herramientas de diseño de interfaces gráficas se encuadran dentro de las denominadas **RAD (Rapid Application Development).**

En nuestro caso destacaremos **Glade**, de software libre, se considera una herramienta de diseño de interfaces pura ya que ésta es su única función a diferencia de otras como Visual Basic que además permite codificar el código. Posteriormente podemos enlazarla con aquellos lenguajes de programación que deseemos. Otras como **Gambas, Mono, Kdevelop, Lazarus y Anjuta,** también de software libre, incluyen además las herramientas necesarias para elaborar el código. En Java podemos citar a **Eclipse y NetBeans** ambas de software libre aunque estos son compatibles con otros lenguajes añadiéndoles los plugins correspondientes.

De software privativo tenemos a **Visual Studio, Velneo, Jcreator, Delphi y Oracle** como las más destacadas y todas ellas son **IDE** que incluyen el software de desarrollo de código además de la interfaz de usuario.

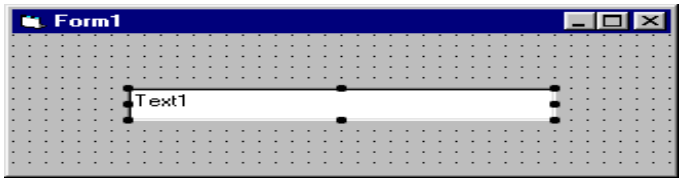
➔ Controis: propiedades

Todos los controles de los que disponen los diferentes **Entorno de Diseño de Interfaces** disponen de una serie de propiedades las cuales podemos cambiar al incluirlos en las aplicaciones. Ejemplos de propiedades son el color, el tipo de letra, el nombre, el texto, etc... En las prácticas de clase veremos como utilizarlas.

A continuación mencionaremos los controles más generales y usuales que aparecen en los interfaces de usuario:

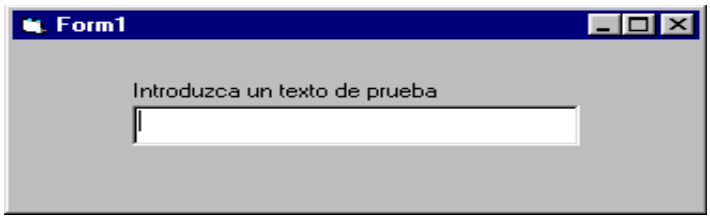
- **Textbox (entry en Glade)**

Mediante este control podremos realizar tanto la entrada como la salida de datos en nuestras aplicaciones.



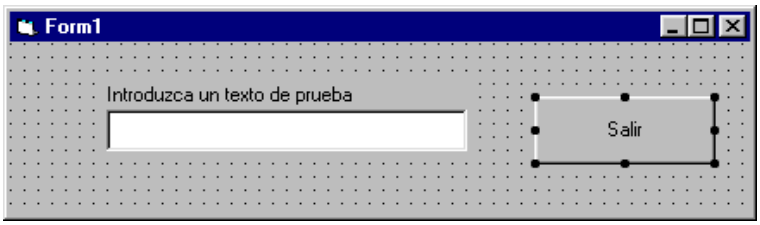
- **Label**

Este control es también uno de los más utilizados, aunque su utilidad queda restringida a la visualización de datos en el mismo, no permitiendo la introducción de datos por parte del usuario.



- **CommandButton o Button**

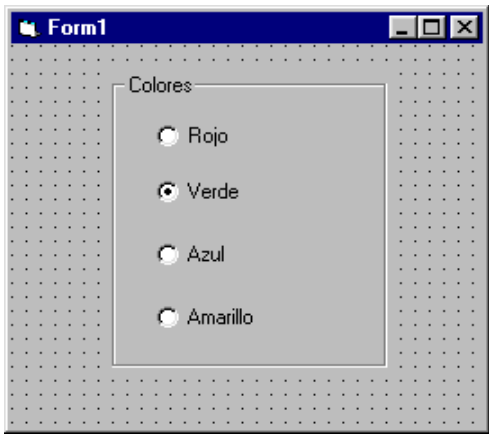
Este control es el típico botón que aparece en todas las aplicaciones y que al hacer click sobre él nos permite realizar alguna operación concreta, normalmente Aceptar o Cancelar. Aunque según el código que le asociemos podremos realizar las operaciones que queramos.



- **OptionButton o Option**

Este control nos permite elegir una opción entre varias de las que se nos plantean. Cada opción será un control optionbutton diferente. De todas las opciones que se nos ofrece, en este caso los 4 colores, sólo podremos activar una. Si activamos cualquier otra opción, se desactivará automáticamente la última que teníamos activada.

Facilita la introducción de datos por parte del usuario:



El marco que está alrededor de los 4 controles optionbutton se trata del control *Frame*  , es opcional, aunque es conviente colocarlo siempre que hagamos uso de las opciones. No sólo por motivos estéticos sino porque de esta manera podremos establecer grupos de controles optionbutton independientes en los que en cada grupo sólo pueda haber una opcioón activada a la vez. También, al mover el marco se moverán los controles incluidos en él facilitándonos las modificaciones. En Glade sería los Container.

- **Combobox**

Permite la selección de un lista que aparece al desplegarla.

Hay infinidad de controles más dependiendo del entorno de desarrollo, pero si analizamos cualquier aplicación en su mayoría la presencia de estos es mayoritaria. A lo largo de las prácticas veremos más.

➔ Eventos: controladores

La programación orientada a eventos o evento de programación basado en un paradigma de

programación en el que el flujo del programa es d**eterminada por los acontecimientos**-por ejemplo, salidas de los sensores o las acciones del usuario o mensajes de otros programas o hilos.

A pesar de que el programa en sí no puede ser predominantemente de eventos determinados eventos controlados, anormales, como un error de hardware, desbordamiento o "cheques del programa" puede ocurrir que posiblemente impide su posterior procesamiento para ello tenemos **los manejadores de excepciones** para dar una solución a la aplicación dando una salida que impida su bloqueo (un mensaje de vuelta atrás, un botón de salida....)

El primer paso en el desarrollo de un programa orientado a eventos es escribir una serie de subrutinas, o métodos, **llamados rutinas de controlador de eventos**. Estas rutinas manejan los eventos a los que el programa principal responderá. Por ejemplo, un solo botón izquierdo del ratón y haga clic en un botón de comando en un programa de interfaz gráfica de usuario puede activar una rutina que se abrirá otra ventana, guardar los datos en una base de datos o salir de la aplicación. Muchos entornos de programación de hoy en día proporcionan al programador plantillas de eventos para que el programador sólo necesita para suministrar el código de evento.

El segundo paso es **enlazar controladores de eventos a los eventos** para que el funcionamiento correcto se llama cuando tiene lugar el evento. Editores gráficos se combinan los dos primeros pasos: Haga doble clic en un botón, y el editor crea un controlador de eventos asociados con el usuario hace clic en el botón y se abre una ventana de texto para que puedas editar el controlador de eventos.

El tercer paso en el desarrollo de un programa orientado a eventos es e**scribir el bucle principal.** Esta es una función que comprueba la ocurrencia de los hechos, y luego llama al controlador de eventos correspondiente a procesarlo. La mayoría de los entornos de programación orientados a eventos ya ofrecen este bucle principal, por lo que no serán necesarios específicamente por el programador de la aplicación.

Programación controlada por eventos es a**mpliamente utilizado en las interfaces gráficas** de usuario, ya que ha sido adoptado por la mayoría de widget toolkits comerciales como el modelo para la interacción.

Como hemos visto entonces un **evento** es un **suceso que ocurre en el entorno donde se ejecuta una aplicación** y para su atención se debe programar en ella un **método o controlador** que controle las acciones necesarias para responderle a cada tipo de evento que se necesite. A diferencia de la programación secuencial, batch por lotes o tipo consola (en la que el programador

exige una secuencia estricta de aparición de los eventos), **no se sabe cual será el orden de aparición** de ellos pues normalmente son el usuario o el sistema operativo quienes los generan según su intención y necesidades. La aplicación debe adoptar, entonces, la forma de intérprete o de máquina de estados para que esté atenta a reaccionar según sea el caso.

El sistema operativo ya existe pero debe invocar un método (enviar un mensaje a la aplicación)que solo ahora la persona programa. Por ello el programador debe hacer dos cosa:

- **Elaborar el método o algoritmo** (mensaje que la aplicación entenderá) que se ejecutará ante la aparición de un tipo de evento.
- Inscribir ese método como un **controlador o manejador** dentro de la aplicación para que el sistema operativo sepa qué invocar.

Para lo primero debe seguirse un protocolo o firma del método que case con lo que ya existe programado en el Sistema Operativo. Constará de un nombre y los parámetros de entrada pues él enviará información a la aplicación sobre el control en el que ocurrió el evento y un paquete con datos relevantes a ese tipo de evento. La forma de los métodos manejadores de eventos será, entonces:

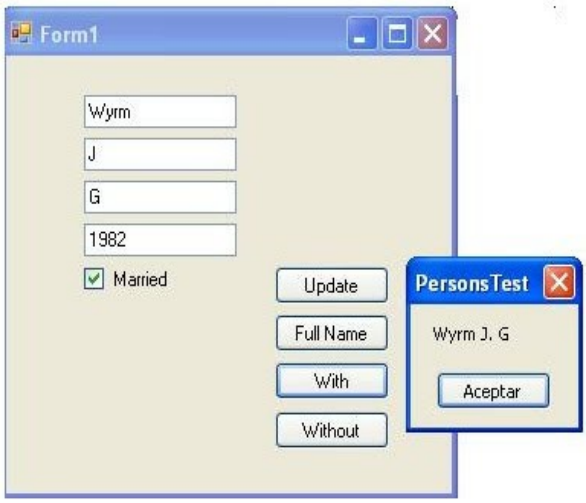
```
nombre_del_método(demeNC, deme detalles es un TipoDeArgumentosDeEvento)
 {
 /* ...aquí irá la lógica del control y puede utilizar la información enviada en detalles por el Sistema Operativo....(el TipoDeArgumentosDeEvento que se recibe varía dependiendo del tipo de evento) */
}
```

Para lo segundo, debe instruir al Sistema Operativo sobre el control donde interesa la ocurrencia del evento, el tipo de evento y el método que lo atenderá, mediante el uso del método Léxico llamado manejador, así por ejemplo cuando exista el botón *BOT* en la aplicación *Aplicación* y se desee atender el evento *Click* con el algoritmo *MiAlgoritmo* se programará en su constructor:

Hay muchos eventos reconocidos por el sistema operativo y quizás los más utilizados sean:

- Paint Pintura se genera cada vez que el S.O. detecta la necesidad de restaurar la imagen de un formulario
- KeyDown: Tecla Presionada por el usuario (o simulada por algún programa)
- MouseDown: Un botón del ratón ha sido presionado (o simulado).
- Tick: Un temporizador ha generado un aviso o alarma
- MouseUp: Un botón del ratón ha sido liberado (o simulado).
- MouseMove: El ratón ha sido movido (o simulado).

Aunque hay muchos más. Resumiendos un controlador de eventos es un **procedimiento del código** que determina qué acciones se ejecutan cuando se produce un evento, como cuando un usuario hace clic en un botón o una cola de mensajes recibe otro mensaje. Cuando se produce un evento, se ejecuta el controlador o controladores de eventos que reciben dicho evento. Los eventos pueden asignarse a múltiples controladores, y los métodos que controlan determinados eventos pueden modificarse de manera dinámica.



➔ Edición y depuración del documento XML.

La **validación XML** *(eXtensible Markup Language)* es la comprobación de que un documento en lenguaje XML está bien formado y se ajusta a una estructura definida. Un documento bien formado sigue las reglas básicas de XML establecidas para el diseño de documentos. Un documento válido además respeta las normas dictadas por su **DTD (definición de tipo de documento) o esquema utilizado.**

La validación se encarga de verificar:

- **La corrección de los datos:** aunque validar contra un esquema no garantiza al 100% que los datos son correctos, nos permite detectar formatos nulos o valores fuera de rango y por tanto incorrectos.
- **La integridad de los datos:** al validar, se comprueba que toda la información obligatoria está presente en el documento.
- **El entendimiento compartido de los datos:** a través de la validación se comprueba que el emisor y receptor perciban el documento de la misma manera, que lo interpreten igual.

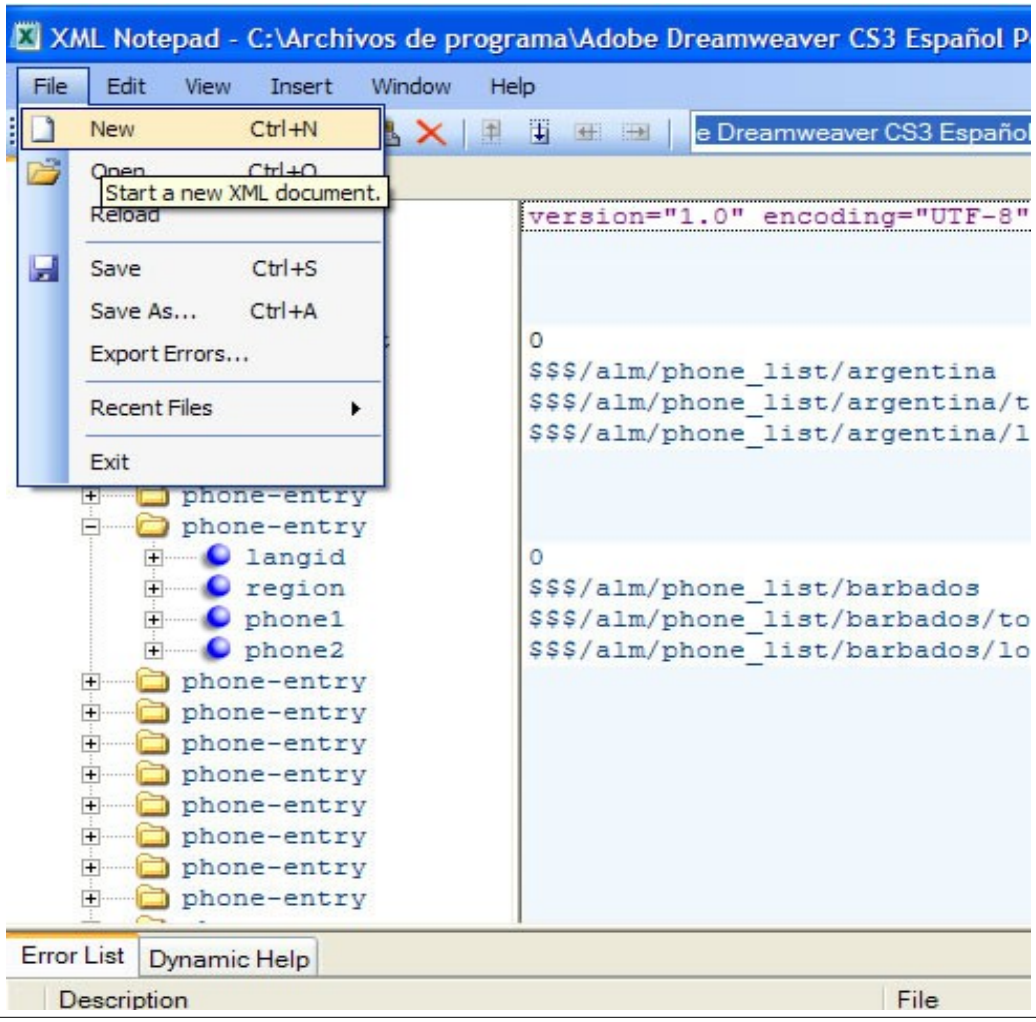
La creación manual de documentos XML e incluso su manipulación pueden introducir todo tipo de errores, tipográficos, sintácticos y de contenido. Existen editores de XML que facilitan la tarea de crear documentos válidos y bien formados, ya que pueden advertir de los errores básicos

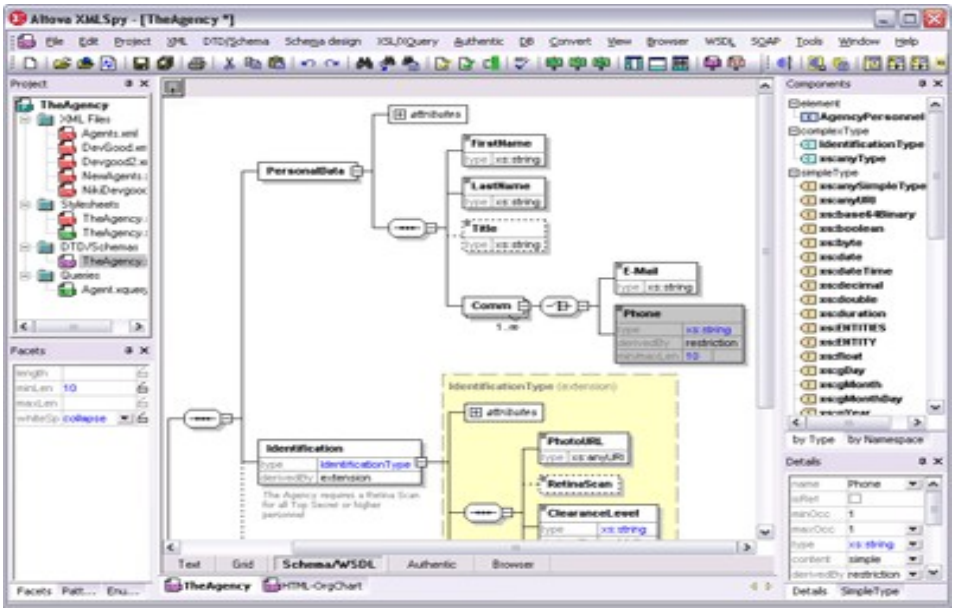
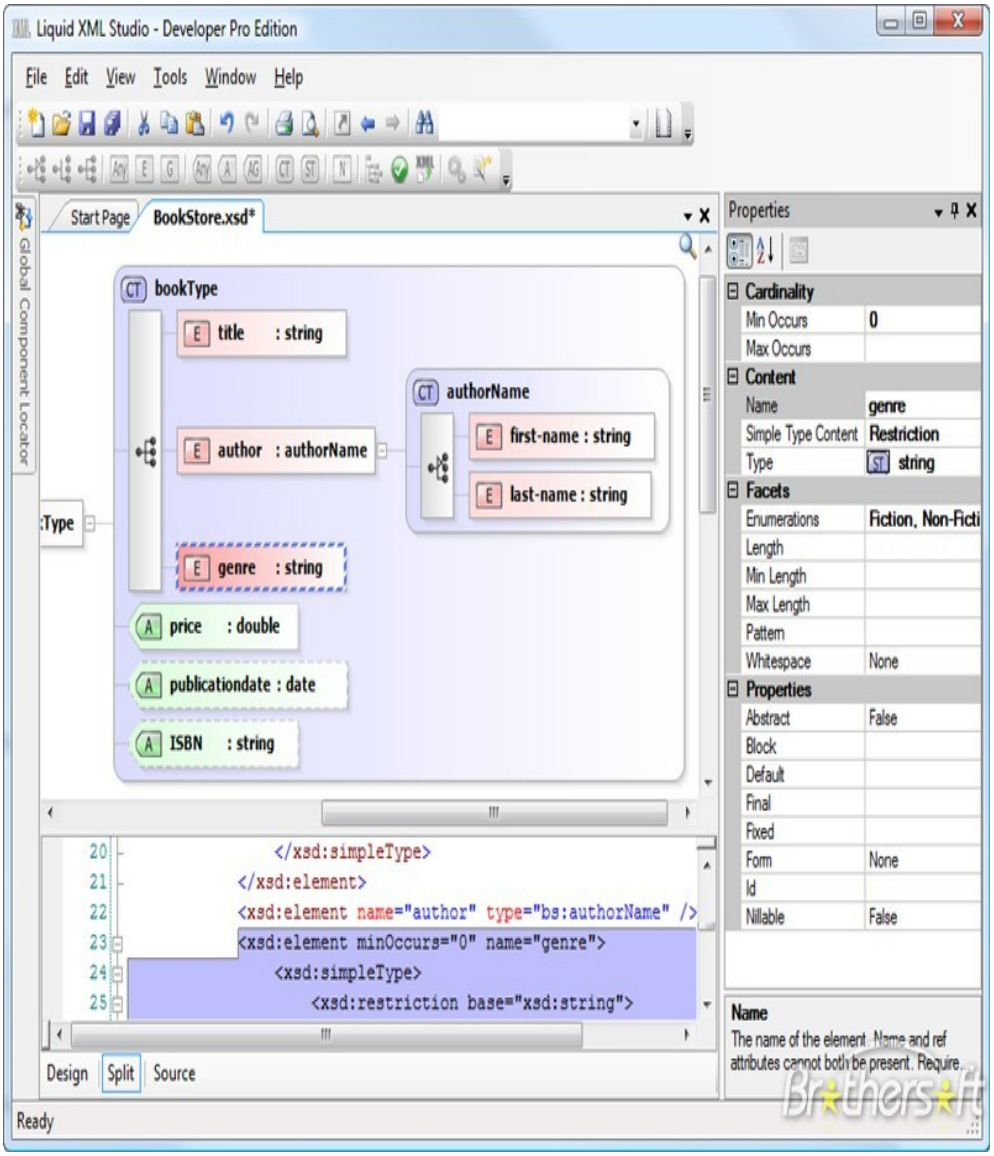
cometidos e incluso escribir automáticamente la sintaxis más sencilla necesaria.

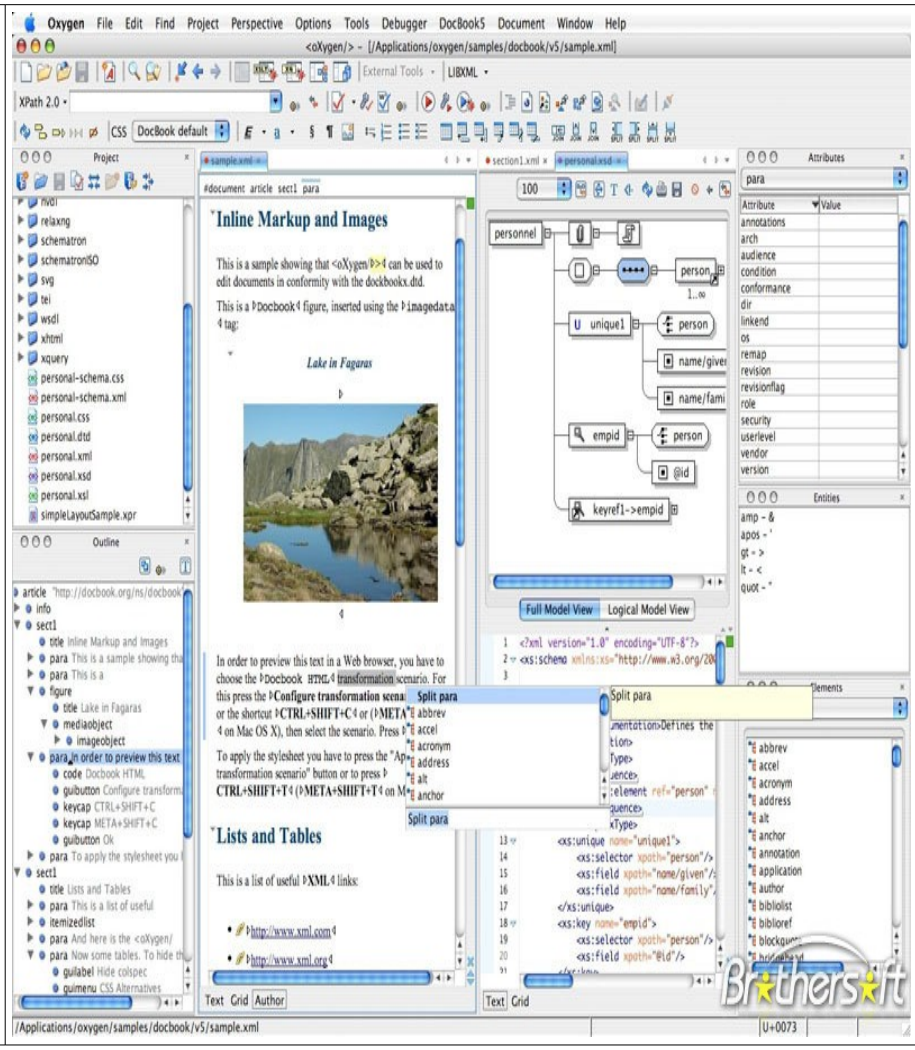
Cuando necesitamos obtener un documento válido, el editor XML ha de ser capaz de:

- Leer la DTD del documento y presentarle una lista desplegable con los elementos disponibles enumerados en la DTD, evitando así la inclusión de algún elemento no definido en el esquema.
- Advertir el olvido de una etiqueta obligatoria e incluso no permitir este tipo de descuidos o errores, no dando por finalizado el documento si existen errores de este tipo.

Algunos editores:

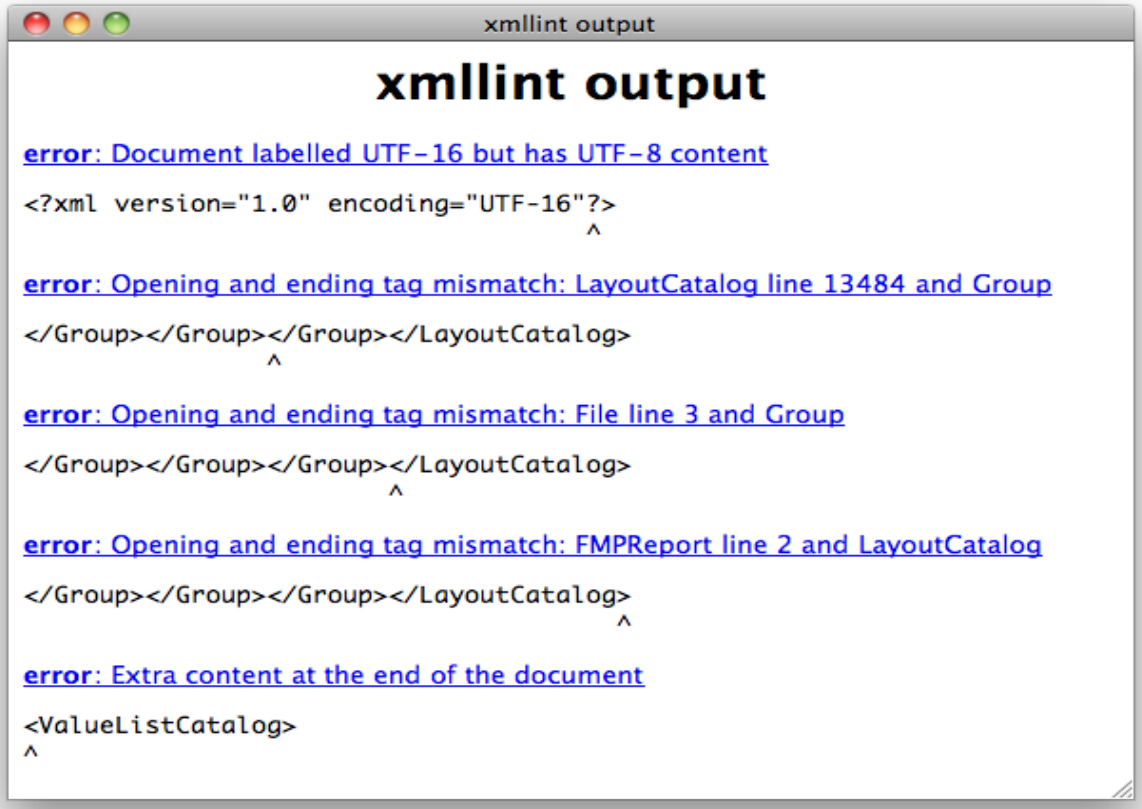
XML Notepad	

XML Spy de Altova	
Liquid XML	

OXigen XML Editor	

A partir de la propia búsqueda del programador de aquellos errores en un documento XML, cabe la posibilidad de usar herramientas para la depuración y optimización del código. Los editores de XML citados como Altova, Macromedia, Exchanger XML Editor... y cualquier editor XML medianamente profesional puede depurar el código XML.

- **xmllint** es una aplicación, analizador de XM basada en la consola, diseñada para imitar xmllint en Windows. Actualmente sólo tiene el formato xmllint implementado. El programa xmllint analiza uno o más archivos XML, especificados en la línea de comandos como archivo_xml. Imprime diversos tipos de salida, dependiendo de las opciones seleccionadas. Es útil para detectar errores tanto en el analizador XML propio como en el código XML.
- **XML Cleaner** es un filtro de validación para ficheros XML, muy similar a xmllint. Nos permite eliminar declaraciones *namespace* superfluas de un fichero XML, nos da elección entre varios tipos de formularios y etiquetas vacias, ofrece un validador XML, soporta Unicode y permite eliminar los comentarios y espacios en blanco.



Para finalizar comentar que los documentos XML **se procesan a través de analizadores,** aplicaciones que leen el documento, lo interpretan y generan una salida basada en sus contenidos y en la marca utilizada para su descripción. El resultado se muestra en un dispositivo de visualización, como una ventana de navegación o una impresora. Los procesadores hacen posible la presentación y distribución de documentos XML.

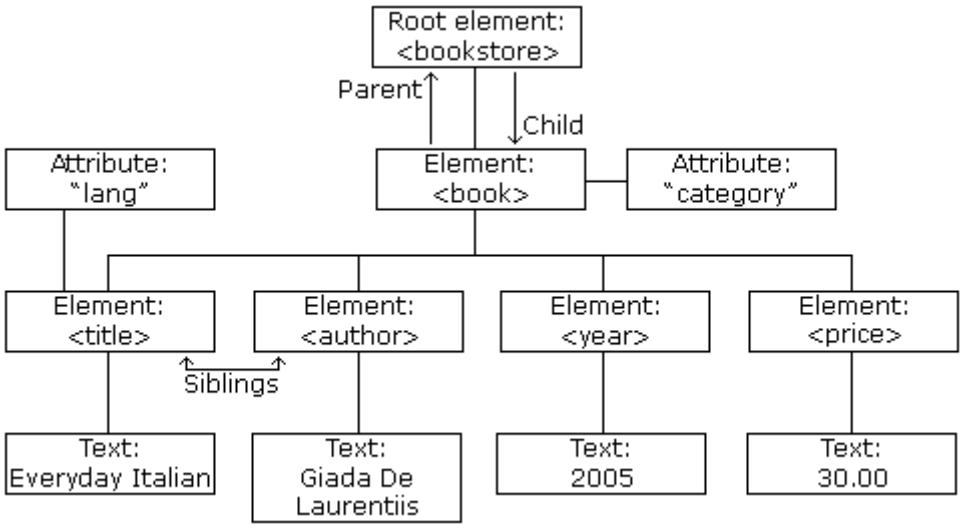
Una aplicación que consume información XML debe leer un fichero de texto codificado según dicho estándar, cagar la información en memoria y, desde allí, procesar esos datos para obtener unos resultados (que posiblemente también almacenara de forma persistente en otro fichero XML).

El proceso anterior se enmarca dentro de lo que se conoce en informática como **“parsing” o análisis léxico-sintáctico**, y los programas que lo llevan a cabo se denominan “parsers” o **analizadores (léxico-sintácticos)**. Más específicamente podemos decir que el “parsing XML” es el proceso mediante el cual se lee y se analiza un documento XML para comprobar que está bien

formado para, posteriormente, pasar el contenido de ese documento a una aplicación cliente que necesite consumir dicha información.

Hay dos tipos de analizadores para documentos XML**: analizadores dirigidos por la estructura (parsers DOM); y analizadores orientados a eventos (parsers SAX).**

- **DOM:** XML es un metalenguaje de **estructura jerárquica**: las marcas dentro de un documento XML tiene una relación padre-hijo estricta. Esta estructura lleva a que la representación natural para documentos de este tipo sea la denominada (en informática) **estructura de "árbol"**. DOM ofrece, a través de sus interfaces, una visión abstracta de esa estructura de árbol. Cada uno de los elementos constructivos del documento XML viene representado por objetos de **tipo "nodo"** basado en los principios del diseño orientado a objetos. Estos nodos serán: elementos, atributos, comentarios, instrucciones de procesamiento, etc.



- **SAX:** El API SAX no especifica como ha de ser la implementación del *parser*, sino como ha de comportarse, es decir, especifica interfaces de programación y no clases. Por lo tanto, SAX no es ningún programa concreto, sino una especificación abstracta que envuelve (mediante la técnica de "*layering*" habitual en POO) a implementaciones de *parsers* XML distintos.

  En su núcleo SAX está compuesto por dos interfaces: el XMLReader que representa al *parser*; y el ContentHandler que recibe datos del *parser*. Estos dos interfaces son suficientes para realizar el 90% de la funcionalidad que podemos necesitar de SAX. En el presente curso vamos a ver las operaciones básicas de XMLReader y con un poco más de detalle el interfaz ContentHandler.

➔ Xeración de código para diferentes plataformas.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Cuando nos referimos a plataformas incluímos Linux, Android para móviles, Appel en sus diferentes dispositivos, Microsfot en sus desarrollos Visual Studio o XPS. También se puede usar en bases de datos, editores de texto, hojas de cálculo, dispositivos móviles, portátiles y casi cualquier cosa imaginable.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/principal"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical">
   <TextView>
      android:id="@+id/etiqueta1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Esto es una etiqueta de texto">
   </TextView>
   <Button>
      android:id="@+id/boton1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Esto es un botón">
   </Button>
</LinearLayout>
```

Código XML para Interfaz en Android

XML es la base para la creación de interfaces gráficas de muchos generadores de código o IDE como Visual Studio, NetBeans y Eclipse para Java y en el caso práctico del proyecto en el desarrollo de interfaces con Glade+Phyton.