-

TEMA 2

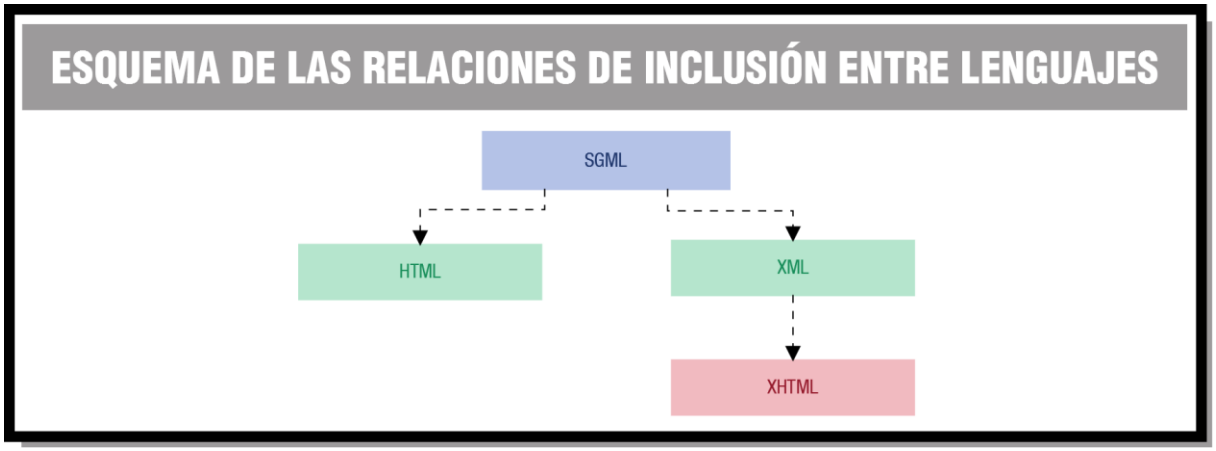
ELABORACIÓN DE INTERFACES MEDIANTE DOCUMENTOS XML

Desenvolvemento de Aplicacións Multimedia

Lenguajes basados en XML

¿Qué es XML?

XML (eXtensible Markup Language o Lenguaje de Marcado eXtensible) es un lenguaje basado en **texto plano**[1] para la definición de documentos, y que permite representar información estructurada de gran variedad de documentos.



En la imagen se ve un diagrama con un cuadro principal con el nombre del primer lenguaje de marcas que se creó en el contexto de Internet, SGML. Del mismo nacen los lenguajes HTML y XML, entendibles a partir de SGML. Podemos considerar a XHTML como hijo de XML en tanto que cumple sus características pero, a la vez, no deja de ser una variante de HTML, por lo que incluso podríamos hablar, hasta cierto punto, de **herencia múltiple**.

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

Orígenes

Hacer independiente la estructura de un documento ante la constante evolución de las herramientas que se usan para visualizarlo ha sido, desde hace tiempo, foco de interés de los desarrolladores y desarrolladoras de aplicaciones informáticas. En los años

[1] Archivo de texto que se compone de exclusivamente de caracteres sin aplicar formato alguno.

sesenta, Charles F. Goldfab, por encargo de **IBM**[2] desarrolló el lenguaje GML (*Generalized Markup Language*), cuyo objetivo era describir la estructura de los documentos de forma que el resultado no dependiese de una determinada plataforma o una aplicación específica. De la evolución de GML surgió SGML que se convirtió en el estándar internacional ISO 8879. Sin embargo tuvo difusión tan sólo en medios académicos y de la administración, y no alcanzó demasiado éxito entre los usuarios medios debido a su compleja estructura.

El hito que supuso la amplia difusión de las tecnologías de la comunicación y la circulación de documentos electrónicos entre cualquier punto del mundo fue la creación de la World Wide Web (Tim Berners-Lee, en 1990), y, asociado a ella, la aparición de **HTML**[3], lenguaje de descripción de documentos basado en SGML para la Web.

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

- Si en un documento HTML se añade el siguiente código: `El padre del lenguaje <b>GML</b> fue <b>Charles F. Golfab</b>, que lo desarrolló por encargo de <b>IBM</b>.`
- Conseguiré algo parecido a esto: El padre del lenguaje **GML** fue **Charles F. Golfab**, que lo desarrolló por encargo de **IBM**.

Ya que encerrar un texto entre `<b>` y `</b>` equivale a aplicar el formato negrita. Elementos de HTML tales como `<b>` y `</>` se denominan **etiquetas**[4], teniendo cada etiqueta un significado distinto en el diseño del documento final (o en su estructura).

XML, sin embargo, **no incluye ninguna orientación al diseño,** siendo puramente un lenguaje estructural cuyo objetivo es definir cómo se organiza la información en un documento. El diseño se definirá después, haciéndose independiente de la plataforma e incluso de la aplicación con la que se va a visualizar (Web, impreso, como documento de texto y más). Por ejemplo, si definimos una página web con XML podremos verla fácilmente en un navegador web, en un teléfono móvil o incluso en un lector Braille, sin modificar el documento de base. Lo que cambia es la forma en que se interpreta el contenido.

---

[2] International Business Machines. Empresa multinacional estadounidense que se dedica Al comercio de soluciones informáticas en el ámbito del hardware y el software.

[3] HyperText Markup Language o Lenguaje de Marcado para HiperTexto. Lenguaje para la creación de páginas web estáticas.

[4] Código en forma de pareja de elementos <nombre_etiqueta>..</nombre_etiqueta> que se emplea en lenguajes XML para hacer referencia a objetos o componentes de objetos que queremos definir.

Reflexiona

XML no incluye ninguna orientación al diseño. Esto quiere decir que no vamos a hacer referencia a ningún aspecto relativo a la aplicación final en la que visualicemos el documento. Entonces ¿cómo crees que podemos usar el mismo archivo para ver en un navegador que en un teléfono móvil? ¿Qué necesitaremos para visualizar ese contenido de formas diferentes?

Efectivamente, los archivos XML se pueden visualizar en diferentes plataformas, como navegadores web, teléfonos móviles, etc. Para salvar las diferencias se debe acompañar de un archivo auxiliar, que interpretará cómo debe visualizarse el contenido XML en cada dispositivo. Este archivo se denomina XSL, y contiene la definición de cómo se verán los elementos XML en un dispositivo concreto.

Para saber más

Puedes encontrar un completo artículo sobre XML en este enlace:

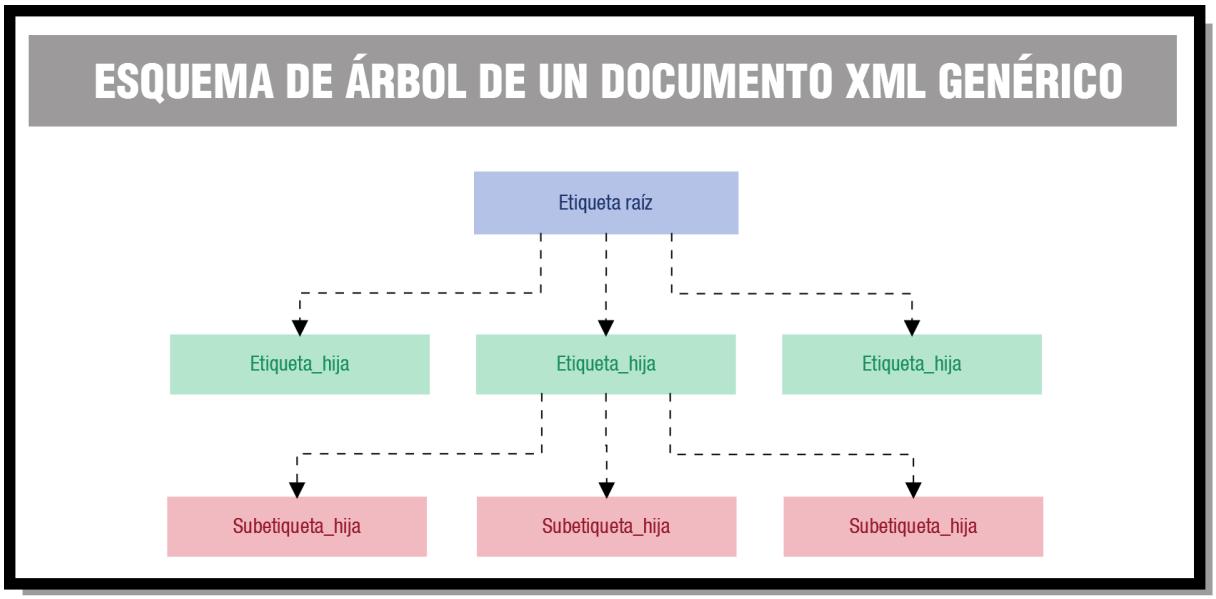
Artículo sobre XML

Vídeo donde se explican algunas características interesantes de XML.

Vídeo

Etiquetas

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



Las etiquetas se delimitan, como vimos anteriormente, mediante **ángulos**[5], e identifican el contenido que delimitan. Pueden poseer atributos adicionales, siguiendo una estructura semejante a lo siguiente:

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

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

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

Estructura de árbol de un documento XML genérico

Estructura XML equivalente a la vista en la imagen anterior.

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

[5] Los símbolos matemáticos de menor (<) y mayor (>).

Por ejemplo, si queremos usar XML para almacenar la información de nuestra agenda de contactos podríamos definir una etiqueta llamada contacto para cada entrada de la agenda y, dentro de los contactos, otras etiquetas para el nombre, teléfono, fecha de nacimiento, grupo, etc. Si tengo un amigo que se llama Javier, cuyo número de teléfono es 637059874 y que nació el 14 de diciembre de 1982, almacenaría la información referente a él en mi agenda de la siguiente manera:

```
<agenda>
  <contacto>
    <nombre>Javier</nombre
    <teléfono>637059874</teléfono>
    <fecha_nacimiento>14-12-1982</fecha_nacimiento>
    <grupo>Amigos</grupo>
  </contacto>
</agenda>
```

Donde la etiqueta raíz sería **agenda**, **contacto** sería una etiqueta hija y **nombre**, **teléfono**, **fecha_nacimiento** y **grupo** funcionarían como subetiquetas.

Atributos y valores

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

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

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

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

Se ven en primer plano unas manos que están manejando una agenda electrónica, y de fondo un ordenador portátil.

```
<agenda>
  <contacto>
    <nombre>Javier</nombre
    <teléfono tipo=”móvil”>637059874</teléfono>
    <fecha_nacimiento>14-12-1982</fecha_nacimiento>
    <grupo>Amigos</grupo>
  </contacto>
</agenda>
```

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

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

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

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

Lenguajes de descripción de interfaces basados en XML

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

- En primer lugar se debe definir lo que se espera de la interfaz, qué requisitos debe cubrir y el entorno en el que se integra.
- También se debe tener en cuenta la variedad de herramientas y lenguajes de programación que exigen un elevado nivel de conocimientos.
- Por otra parte necesitamos interfaces (así como las aplicaciones subyacentes) para múltiples contextos de uso, que dependerán del usuario (por su idioma, preferencias o posibles discapacidades), de la plataforma (ordenadores personales, móviles, miniportátiles, pantallas, etc.), del modo de presentación de la interfaz (gráfica o texto) y del dispositivo final (pantallas de ordenador, pizarras digitales, proyectores, etc.).

Proceso de elaboración de las interfaces

La primera posibilidad para desarrollar una interfaz suele ser emplear el mismo lenguaje, normalmente de alto nivel, que se usa para implementar la funcionalidad de la aplicación (como Java, C#, etc.). Ello tiene como ventaja el hecho de que no es preciso adquirir nuevos conocimientos, así como de que la creación de la interfaz se integra en el proceso de desarrollo de la aplicación y, normalmente, no es necesario trabajar con herramientas diferentes, ni en la programación, ni posteriormente a la hora de **compilar**[6] o ejecutar el programa. Sin embargo, tiene como principal desventaja la dependencia de la plataforma, del dispositivo y del propio lenguaje. Básicamente, la

[6] Proceso de traducción del código fuente de una aplicación a código que un ordenador puede ejecutar. Al código resultante de compilar un código fuente se le denomina código objeto.

portabilidad de las interfaces creadas de esta manera estará limitada a la que proporcione el propio lenguaje.

La creación de interfaces de usuario usando **lenguajes de descripción**[7] basados en XML pretende solventar esto, proporcionando un medio que permita construir interfaces mediante descripciones de alto nivel relativos a los distintos aspectos de la propia interfaz (estructura y comportamiento); de modo que a partir de la descripción se pueda generar de forma automatizada la interfaz de usuario final. Además, este proceso permite centrar el desarrollo en las necesidades del usuario, puesto que no se realiza siguiendo las directrices marcadas por el lenguaje.

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

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

Reflexiona

El uso de tecnologías XML permite separar completamente la creación de la interfaz de la programación de la aplicación subyacente. ¿Qué ventajas crees que aporta esto? ¿Puede mejorar el procedimiento de desarrollo de aplicaciones con interfaces gráficas complejas?

De hecho, así es. Utilizar tecnologías XML para la creación de interfaces tiene como principal ventaja que, después de planificar adecuadamente qué esperamos de la aplicación y qué información debe proporcionarle la interfaz, el equipo puede dividirse totalmente el trabajo de desarrollo, ya que ambas tareas pueden realizarse completamente por separado con el consiguiente ahorro de tiempo. Además, las modificaciones que se pueden realizar influyen mucho menos en la aplicación global, y suelen afectan tan solo al ámbito que corresponde. Por otra parte, al utilizar lenguajes y herramientas específicas, normalmente se consiguen interfaces más elaboradas, ya que se ponen a nuestra disposición elementos concretos de dicho ámbito de la programación.

XUL

Es un lenguaje de marcas basado en XML para definir interfaces de usuario complejas, en las que podremos insertar los elementos habituales en un formulario, pero

[7] Lenguaje cuya función no es crear programas sino describir cómo se representan los datos de un documento.

que también dispone de otros más complejos, como menús, barras de herramientas, estructuras jerárquicas o teclas de acceso directo. Así mismo, permite colocar los contenidos de la interfaz usando distribuciones complejas.

Este lenguaje se utiliza específicamente para desarrollar aplicaciones en red. Los documentos escritos en este lenguaje podrán visualizarse en los navegadores Mozilla y Netscape, de hecho la interfaz de ambos está desarrollada en XUL. Ambos navegadores funcionan sobre el motor de **renderizado**[8] Gecko, que es la base que necesitamos para visualizar correctamente el contenido de un documento XUL. Gracias a esto, las aplicaciones XUL son multiplataforma y multidispositivo, ya que siempre podremos encontrar un navegador Gecko para la plataforma y dispositivo deseados.

Uno de los principales atractivos de este lenguaje es que ofrece la posibilidad de crear complementos para el navegador Mozilla.

Por contra su principal desventaja es que no se podrá ejecutar en otros navegadores que, como Internet Explorer, no cumplan con esta característica.

Proceso de mapeado: Un documento XUL es interpretado para visualizar su contenido, en la renderización intervienen la definición de los elementos y las hojas de estilo. Es muy parecido a HTML, utiliza hojas de estilo para diseñar el aspecto y JavaScript para implementar el comportamiento de sus elementos. Para definir una interfaz se especifican tres grupos de componentes distintos:

- **Content:** Aquí se encuentran los documentos XUL, que definen el diseño de la interfaz. Incluye las etiquetas y la referencia a los documentos JavaScript.
- **Skin:** Contiene las hojas de estilos (**CSS**[9]) y las imágenes, las cuales definen la apariencia de la interfaz.
- **Locale:** Los documentos **DTD**[10] se encuentran aquí, estos documentos facilitan la localización de páginas XUL.

Cada uno de estos componentes se almacena en un archivo diferente, que puede ser editado.

---

[8] Proceso de generar una imagen a partir de un modelo usando una aplicación informática.

[9] Cascading Style Sheets u Hojas de Estilo en Cascada. Es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores y desarrolladoras el control total sobre el estilo y el formato de sus documentos.

[10] Document Type Definition o Definición de Tipos de Documento. Es un archivo que almacena la definición formal de un tipo de documento y especifica su estructura lógica. En general la utilización del DTD es opcional aunque conveniente. En función de si se utiliza o no DTD los documentos serán 'válidos' o sólo 'bien formados'.

Este es un ejemplo de archivo XUL, puedes encontrar un ejemplo un poco más extendido en este enlace:

Ejemplo de XUL

Para saber más

Página oficial de XUL donde poder encontrar la referencia del lenguaje, ejemplos de aplicaciones. Cabe destacar el enlace a la aplicación XULRunner que te permitirá crear y ejecutar aplicaciones XUL:

Página oficial de XUL

En este enlace encontrarás un pequeño tutorial, muy sencillo de seguir, para ver cómo funciona una aplicación web muy básica escrita en XUL.

Mi primera aplicación XUL

XAML

Es un lenguaje de marcas empleado para la creación de interfaces en el modelo de programación .NET **Framework**[11] de Microsoft. Las aplicaciones creadas podrán ejecutarse en entornos Windows.

Mediante XAML se pueden crear lo que se conoce como **RIA**[12] (*Rich Interface Applications*) que contienen abundante material gráfico. XAML consta de una serie de elementos XML para representar los principales componentes gráficos, así como la distribución, paneles y manejadores de eventos. Puede hacerse programando directamente la interfaz mediante un editor de texto, o mediante el entorno de desarrollo gráfico incluido en la herramienta Expresión Blend. El código asociado a la interfaz es puramente declarativo, es decir, hace referencia tan solo al aspecto visual, pero no añade funcionalidad, salvo ciertas respuestas muy sencillas a interacciones con el usuario.

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

[11] Conjunto de módulos software que permiten la elaboración y desarrollo de otros proyectos software haciendo independiente al programador o programadora los detalles de bajo nivel dependientes de la máquina. Suelen incluir módulos para edición, compilación y enlazado, o para la interpretación de un lenguaje.

[12] Rich Internet Application o Aplicación para Internet Enriquecida. Este tipo de aplicaciones se ejecutan en un servidor web y se visualizan en navegadores web, pero tienen interfaces gráficas con una complejidad similar a las aplicaciones de escritorio.

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

Tienes el código del ejemplo en el siguiente documento:

Documento con ejemplo para XAML

Para saber más

Si quieres conocer algo más sobre XAML te sugerimos una página donde poder encontrar una introducción a XAML:

Página oficial de XAML

Y también puedes visitar la siguiente página donde poder encontrar la referencia del lenguaje y ejemplos de aplicaciones dentro del ámbito del entorno de desarrollo Expression Blend:

Página de la MSDN de XAML

UIML

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

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

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

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

En UIML, una interfaz de usuario es una jerarquía de elementos XML. Cada uno de los componentes de una interfaz (botones, cajas de texto, etiquetas...) son una entidad

XML. Estas entidades son elementos **Part**. Cada uno tiene asociado un elemento Content, que puede ser texto, imagen, etc.

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

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

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

Tienes el código del ejemplo en el siguiente enlace:

Ejemplo de UIML

Para saber más

Aquí tienes un enlace donde poder encontrar la especificación del lenguaje, tutoriales y ejemplos de aplicaciones:

Página oficial de UIML

XIML

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

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

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

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

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

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

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

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

Herramientas para la creación de interfaces multiplataforma

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

Cuando creamos una aplicación con interfaces gráficas, en primer lugar hay que **diseñar la interfaz**, normalmente el **toolkit**[13] contiene los objetos gráficos más empleados, tales como menús, botones, etiquetas, barras de scroll, y campos para entrada de texto que compondrán la interfaz, mientras que el sistema de ventanas provee de procedimientos que permiten dibujar figuras en la pantalla y sirve como medio de entrada de las acciones del usuario. En la imagen puedes apreciar los conjuntos de herramientas enmarcados en rojo (en la imagen cuadro derecho y barra de herramientas superior) y la zona de dibujo en azul (en la imagen cuadro central).

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

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

[13] Conjunto de herramientas, widgets, ventanas, etc. que pueden ser usados para construir una interfaz.

hacer las transformaciones necesarias para visualizar la interfaz. Este proceso puede ser más o menos automático en función de las tecnologías seleccionadas.

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

Presentación de algunas herramientas

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

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

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

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

- Libres:
  - QT Designer.
  - Glade.
- Propietarias:
  - Expression Blend de Microsoft.
  - Flex de Adobe.

Para saber más

Si tienes curiosidad por saber algo más de estas herramientas aquí tienes lo enlaces que te llevarán a sus páginas oficiales:

Página oficial de QT

Página oficial de GLADE

Página oficial de FLEX

Página oficial de Microsoft Expression Blend

Página de MSDN para Expression Blend

Ejemplo de desarrollo de una interfaz básica con QT Designer

QT es un entorno de desarrollo de aplicaciones multiplataforma para la creación de aplicaciones web, de escritorio y para móviles. Consta de un conjunto de herramientas de desarrollo y un API, las bibliotecas QT, escrito en C++.

El **IDE**[14] se denomina QT Creator y dispone de un conjunto de módulos para el desarrollo completo de aplicaciones de diversos tipos, entre ellas nos centraremos en QT Designer que es una herramienta de tipo WYSIWYG para la elaboración de interfaces de usuario complejas basadas en componentes de las bibliotecas QT. Se pueden incluir todo tipo de elementos como menús, paneles, barras de herramientas, y disponerlos usando distribuciones muy completas.

Entre todas las herramientas disponibles para el desarrollo de interfaces basadas en tecnologías XML, se ha optado por QT Designer por varios motivos:

- Está disponible bajo licencia LPGL, aunque combina con una licencia comercial.
- Es una herramienta multiplataforma, estando soportada por las principales plataformas del mercado (Windows, Linux, MACOS) así como otras más emergentes, como Windows Mobile, Symbian, Maemo, o, más reciente, Android.
- Produce código portable.
- Utiliza código intermedio XML para definir las interfaces de las aplicaciones.
- Es posible integrarlo de forma sencilla con las herramientas que estamos usando para realizar las prácticas en este ciclo e involucrar el desarrollo de interfaces en XML dentro de proyectos de mayor envergadura sin necesidad de aprender otras nuevas.

[14] Integrated Development Environment o Entorno de Desarrollo Integrado. Conjunto de herramientas para el desarrollo de programas informáticos que cubren todo el ciclo de vida de la aplicación y se utilizan en conjunto.

Breve historia

QT apareció como biblioteca en 1992 para Windows y X11, creada por Haavard Nord y Eirik Chambe-Eng. Más tarde, en 1994 surge la empresa Troll Tech (antes Quasar Technologies), encargada de su desarrollo. En esa época QT se involucra en el desarrollo del escritorio KDE. No todo el mundo estaba de acuerdo en que una plataforma que no era totalmente libre participara en el proyecto GNU, lo que hizo que paulatinamente fueran surgiendo licencias LGPL para todas las plataformas. En el año 2008 fue adquirida por Nokia, y comenzó su expansión por la plataformas móviles (Windows Mobile, Symbian), liberando en febrero de 2011 la versión Alpha para Android, con lo que se convierte en uno de los entornos de desarrollo más completos con respecto a dispositivos finales cubiertos.

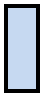
El *binding* QT Jambi

Las bibliotecas QT están desarrolladas en y para el lenguaje C++, sin embargo nosotros vamos a trabajar con Java. Para hacerlo posible utilizaremos un binding que es un traductor que permite a desarrolladores y desarrolladoras de determinados lenguajes utilizar las tecnologías QT, así como a los programadores y programadoras QT exportar su trabajo a otros lenguajes. El **binding**[15] QT/Java, que se lama QT Jambi, permite a los desarrolladores Java utilizar las tecnologías QT mediante una capa intermedia de código estático que transmite las llamadas a la API de Java a la biblioteca compilada QT de C++

Es una taza tipo mug con café de color verde. Lleva pintada una letra Q mayúscula, y una T mayúscula inclinada.

Reflexiona

Los bindings extienden el uso de bibliotecas escritas en un lenguaje determinado a otros lenguajes de programación. El código QT Jambi será Java por completo, sin que el programador tenga que preocuparse de la arquitectura de las clases QT en C++.



[15] Adaptación de una biblioteca para ser usada por un lenguaje para el cual no ha sido creada.

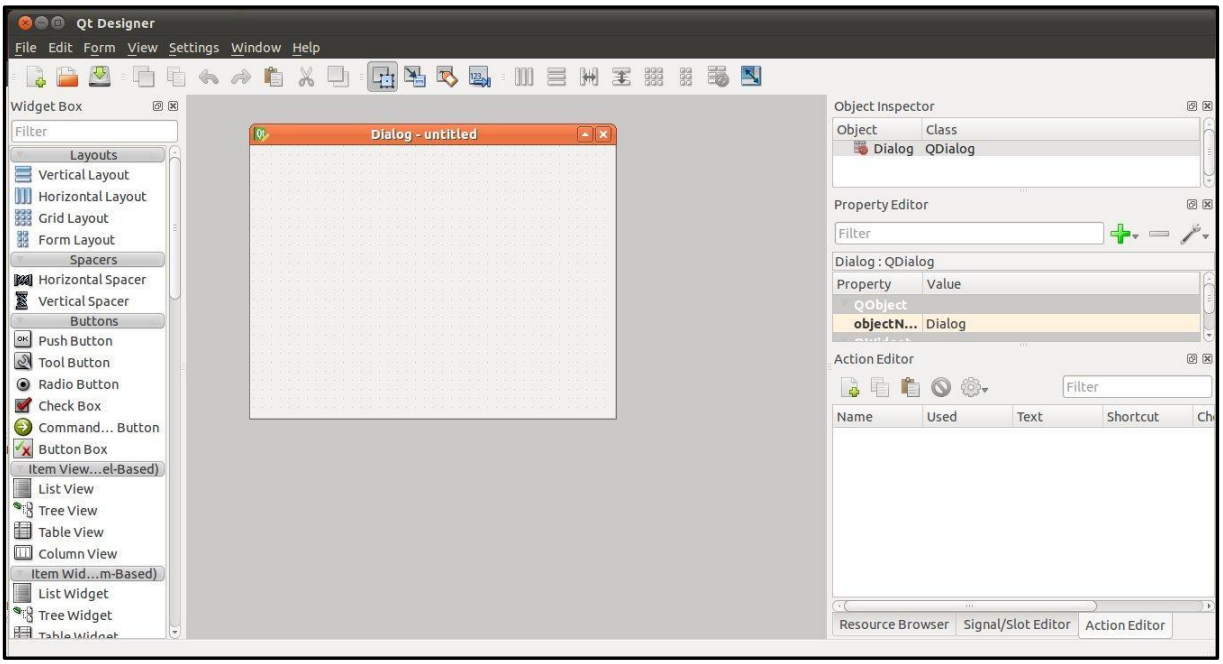
Para saber más

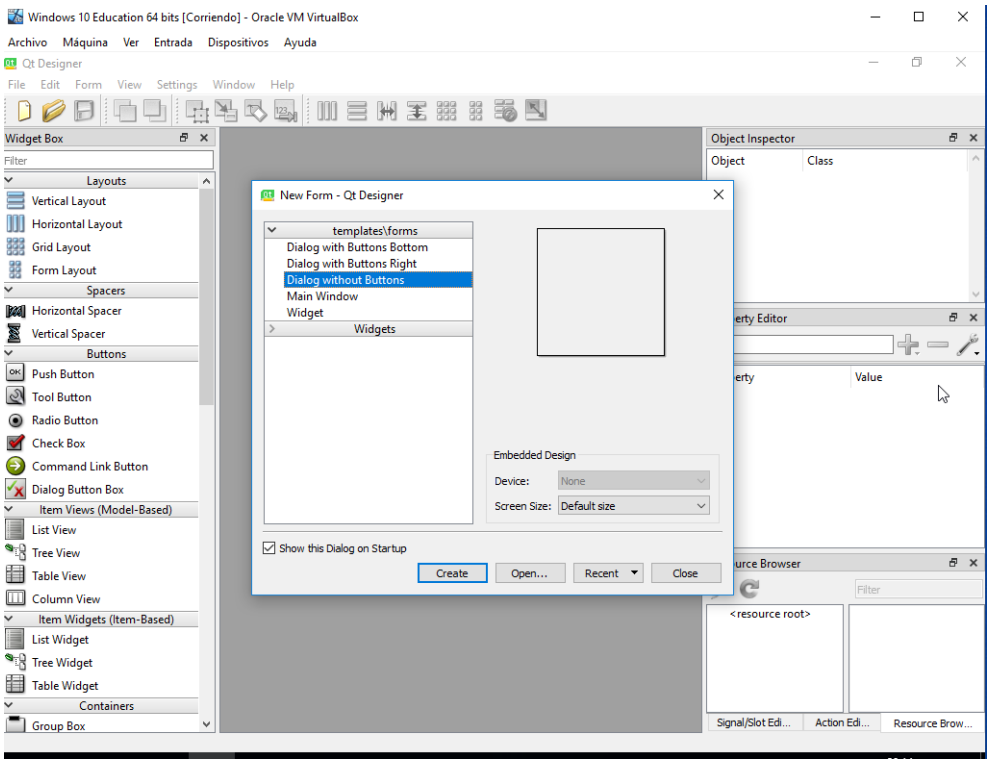
QT es un entorno de desarrollo de aplicaciones multiplataforma en lenguaje C++ muy completo. Se ha usado para desarrollar, entre otras cosas, el escritorio KDE de Ubuntu. Si quieres saber algo más de la herramienta QT Creator y el API en general sigue el enlace a la página principal de QT:

Página principal de QT

Revisión general del entorno

Una vez realizada la instalación de QT Designer lo ejecutamos:





Ejecución desde una instalación en Windows 10

Cuenta en la zona superior con un **menú de acceso** a las diferentes opciones para gestionar archivos, portapapeles, ayuda, etc., así como una **barra de herramientas** justo debajo con atajos a las opciones más comunes:

- Relacionadas con la gestión de archivos: nuevo, abrir y guardar.
- Enviar al fondo o al frente.
- Relacionados con los modos de edición del programa: edición de **widgets**[16], edición de signal-slot, edición de buddies y edición del orden de tabulación, que veremos más adelante.
- Relacionados con los elementos de distribución de objetos en el formulario (**layout**[17]).

Se puede modificar la barra de herramientas desde el menú **View >> Toolbars >> Configure Toolbars**.

[16] Objetos gráfico susceptible de formar parte de una interfaz de usuario. Pueden ser sencillos como botones, cuadros de texto o etiquetas, o complejos como paneles con pestañas o layouts. Las herramientas para construcción de interfaces gráficas incluyen una serie de widgets por defecto, pero el usuario puede generar los suyos propios e incluirlos en sus interfaces.

[17] Composición o diseño. En el ámbito de la interfaces de usuario hace referencia a la distribución espacial de los controles de usuario dentro de la interfaz.

A la izquierda se encuentra el **Widget Box**, que ofrece una selección de widgets estándar Qt, layouts, y otros objetos que se pueden utilizar para crear interfaces de usuario. Los widgets se agrupan en categorías según su uso o características.

A la derecha encontramos un conjunto de paneles que permiten gestionar los widgets incluidos en la interfaz, los más importantes son:

- **Object Inspector (Inspector de objetos):** contiene la estructura jerárquica de widgets que se han añadido al formulario que se está creando.
- **Property Editor (Editor de propiedades):** permite la edición de todas las propiedades del widget seleccionado en el inspector de objetos.
- **Editor de Signal/Slot:** el mecanismo Signal-slot se emplea para asociar comportamientos a los elementos de la interfaz.
- **Editor de acciones:** se utiliza para la gestión de acciones. Las acciones se asocian a los elementos de un menú.
- **Editor de recursos:** permiten gestionar los recursos que intervengan en la aplicación, como archivos con imágenes, por ejemplo.

Descripción del problema

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

Para ilustrar el uso de la herramienta Designer y la inclusión de los archivos de interfaz generados en una aplicación Java vamos a desarrollar una pequeña parte de una aplicación de **gestión de un hotel**, en concreto, haremos una interfaz muy sencilla para añadir una reserva de un cliente. Tendremos que escribir los datos personales del cliente y los datos de la reserva, que incluyen la fecha de entrada y salida, número de personas que se van a alojar en el hotel, tipo de habitación, pudiendo elegir entre doble de uso individual, doble, junior suite y suite, y si la habitación será para fumadores o no fumadores Etc .

No vamos a implementar la funcionalidad subyacente a la interfaz, es decir, añadir la reserva a la base de datos del hotel, esto se deja para otros módulos del ciclo.

Una vez que sabemos lo que queremos conseguir, ¿por dónde empezamos?...

Creación del formulario

Recuerda que el acceso a la herramienta era ejecutando QT **designer** .

Para crear una nueva interfaz, seleccionamos **File >> New**, o directamente pulsamos la combinación **Ctrl+N**. Inicialmente se nos preguntará qué tipo de ventana queremos crear, dándonos a elegir entre:

- Diálogo con botones abajo (Dialog with buttons Bottom).
- Diálogo con botones a la derecha (Dialog with Buttons Right).
- Diálogo sin botones (Dialog without Buttons).
- Ventana principal (Main window).
- Widget
- Los **diálogos** se emplean para crear interfaces para la realización de tareas concretas que impliquen comunicación entre el usuario y la aplicación. Suelen usar botones para permitir la interacción con el usuario y realizar acciones.
- La **ventana principal** se suele usar como interfaz inicial de una aplicación. Incluye un menú y una barra de estado. Se le pueden añadir barras de herramientas.
- La interfaz widget se usa para extender la biblioteca y crear nuevos elementos de interfaz que podrán usarse posteriormente.

Una vez seleccionado el tipo de interfaz se van añadiendo los elementos de la misma seleccionándolos en el **Widget Box** y arrastrándolos a la zona de la interfaz donde se van a localizar.

En nuestro ejemplo crearemos una interfaz de reserva de habitación a partir de un diálogo sin botones. Se utilizará esta opción para ilustrar cómo se programan los botones después. También crearemos una ventana principal con un menú que permita abrir este diálogo.

Ambas interfaces se almacenan en formato XML en sendos archivos de texto plano, con extensión .jui, que podemos almacenar en un directorio de nuestro disco duro. Más tarde, abriendo estos archivos, podemos recuperar las interfaces para seguir modificándolas desde la aplicación QT Designer o desde un editor de textos.

En los siguientes documentos podrás repasar como se añaden los elementos de interfaz al diálogo y a la ventana. Te recomiendo que los leas una vez para observar qué

se pretende conseguir y cómo hacerlo. Luego puedes ir haciendo tú la interfaz al mismo tiempo que ves cómo se hace:

Visitar Anexo I

Modificar la composición de los elementos de la interfaz

Una vez que hemos situado los elementos que nos interesan vamos a modificar alguna de sus características. Principalmente haremos esto cambiando las propiedades de los objetos en el Editor de propiedades, pero también podemos cambiar algunas características haciendo clic con el botón secundario sobre el elemento a modificar.

Definir la distribución de los elementos

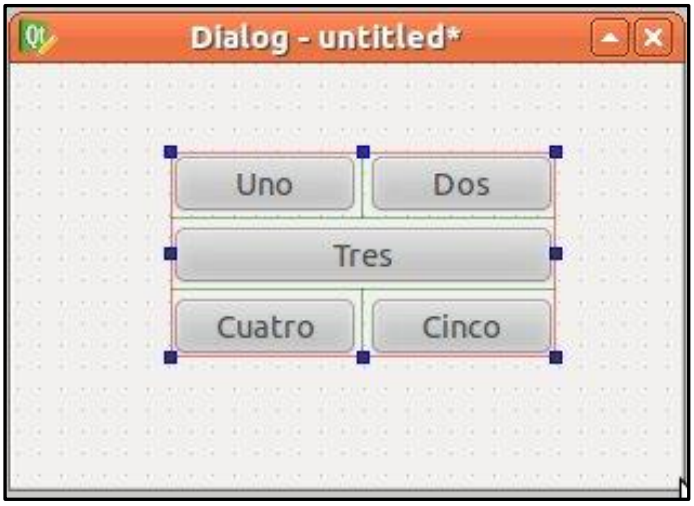
Cuando se desarrolla una interfaz con QT Designer es preciso integrar sus elementos dentro de un layout para garantizar que al visualizarse los objetos se muestren en el lugar indicado. Así mismo nos aseguramos de que los objetos se dimensionarán adecuadamente cuando se redimensione el formulario.

Entendemos por layout o distribución el esquema que definirá la disposición de los elementos de la interfaz.

Podemos organizar los elementos como mejor nos convenga utilizando los siguientes tipos de diseño:

- **Layout horizontal o vertical:** es la distribución más sencilla. Encerrar un conjunto de elementos en un layout horizontal o vertical asegura que se van a mostrar alineados horizontal o verticalmente.

- **Grid Layout:** Permite una distribución compleja de los elementos de la interfaz ya que podemos colocar los objetos en una rejilla, sin embargo puede restar cierta flexibilidad al diseño.



- **Splitter Layout:** Permiten colocar los objetos con una distribución horizontal o vertical, pero dando la posibilidad de escoger el espacio final reservado para cada objeto.

Notas Finales:

No podrá considerarse esta unidad como un manual completo de aprendizaje de QT, sino como ejemplo didáctico de la materia que se pretende trasladar al alumnado.

No vamos a implementar la funcionalidad subyacente a la interfaz, es decir, añadir la reserva a la base de datos del hotel, esto se deja para otros módulos del ciclo.

Anexo I - Creación del diálogo para el alta de clientes

Requisitos de la interfaz

Se pretende crear una interfaz para gestionar las reservas de un hotel. Tenemos que poder escribir los datos personales de la persona que hace la reserva:

- DNI.
- Nombre.
- Dirección.
- Localidad.
- Provincia.

También tenemos que poder incluir los datos de la reserva:

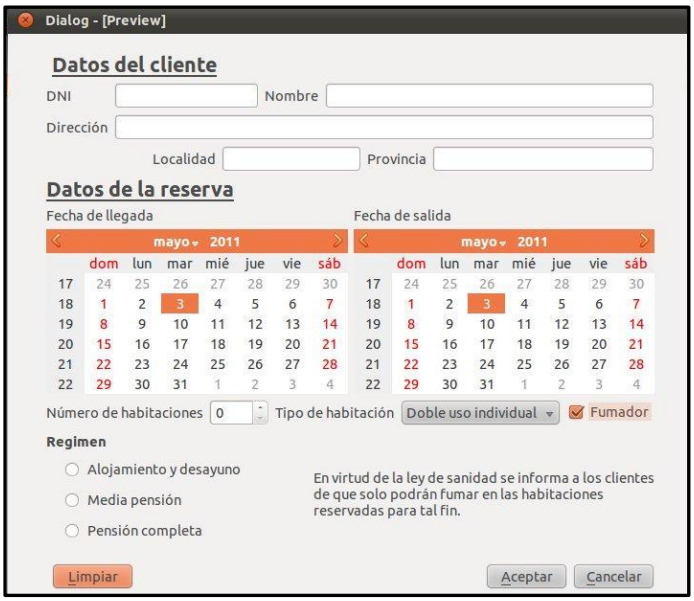
- Fechas de llegada y salida.
- Número de personas.
- Tipo de habitación. El tipo de habitación puede ser:
    - Doble de uso individual.
    - Doble.
    - Junior suite.
    - Suite.
- Si es para fumador o no fumador.
- Régimen de alojamiento. Puede ser:
    - Alojamiento y desayuno.
    - Media pensión.
    - Pensión completa.

Como requerimiento adicional se pide que si la reserva es para fumador se muestre el siguiente mensaje:

"En virtud de la ley de sanidad se informa a los clientes de que solo podrán fumar en las habitaciones reservadas para tal fin."

La interfaz debe contar con las facilidades de atajos de teclado y manejo del tabulador.

Una vez terminada se verá así:

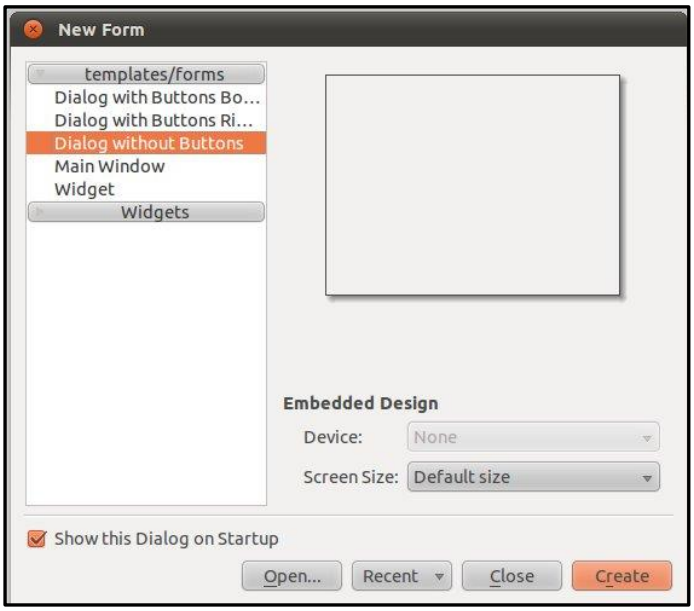


Elaboración de la interfaz

Arrancamos QT Designer:

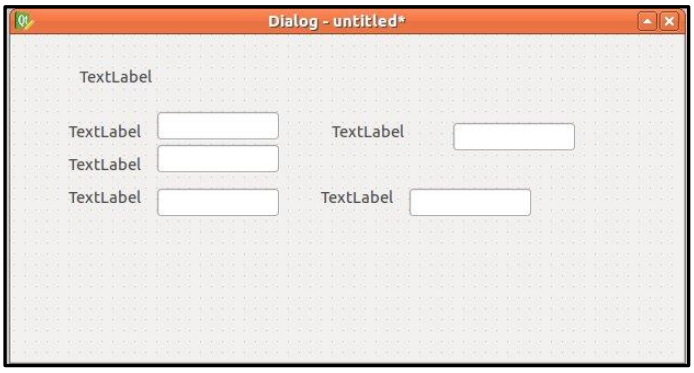
Para empezar nos pide que seleccionemos el tipo de interfaz, en nuestro caso, vamos a crear un diálogo, podemos utilizar el modelo sin botones en la zona inferior.

Como tiene los nombres en inglés luego les cambiaremos las etiquetas.



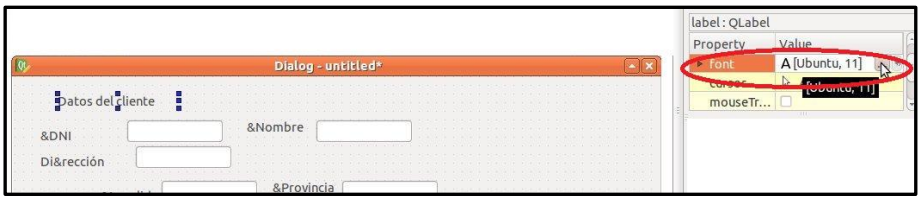
El siguiente paso es añadir los widgets que compondrán la interfaz. En esta herramienta hay que arrastrar el widget desde el panel de la izquierda, hasta situarlo en la zona que queremos. Para insertar los datos del cliente necesitamos etiquetas (**Labels**) y campos de texto (**LineEdits**).

La base para los datos del cliente queda más o menos así:



Si hacemos **doble clic** en una etiqueta podemos editar el texto que aparece, modificamos las que tenemos para que queden como en la imagen de la derecha. Es importante dejar los símbolos & para la edición de buddies, como veremos más adelante.

Podemos modificar el formato de una etiqueta en el panel de propiedades, buscamos la propiedad **font**.



Modificamos el formato de fuente a **Ubuntu, negrita, tamaño 16**, quedando la interfaz así:

Pasamos ahora a añadir los widgets para los datos de la reserva.

- Fechas de llegada y salida >> **Calendar**.
- Número de personas >> **Spinbox**.
- Tipo de habitación >> **ComboBox**.
- Si es para fumador o no fumador >> **Checkbox**.
- Régimen de alojamiento >> Botones de radio (**Radio Button**) dentro de un **GroupBox**.
- El texto con el aviso sobre la prohibición de fumar en el hotel >> **Label**
- Todos los nombres son etiquetas (**Label**)

Para que quepan todos tendremos que agrandar la ventana del diálogo, se agranda arrastrando con el ratón desde los bordes de la ventana. Para mover los botones basta con seleccionarlos con el ratón y arrastrarlos.

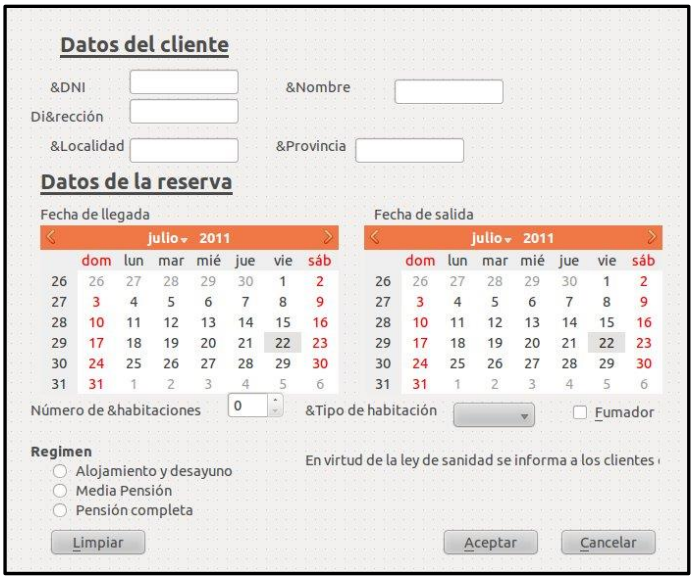
Nota: Cuando añadas el GroupBox y los botones de radio arrastra primero el GroupBox y los botones de radio después para que queden dentro del grupo.

La interfaz queda así:

Ahora modificamos el texto de las etiquetas haciendo doble clic sobre ellas de modo que queden tal y como vemos en la imagen. Modificamos también el formato de la fuente

de la etiqueta "Datos de la reserva" (a Ubuntu 16 negrita) de la misma forma que lo hicimos con "Datos del cliente".

Añadiremos un botón que falta para reiniciar todos los widgets de la interfaz, si es necesario. Para ello añadimos un **PushButton**, haciendo doble clic sobre el podemos cambiar el texto que aparece a &Limpiar. Y dos más para Aceptar y Cancelar y ya tenemos la interfaz diseñada.

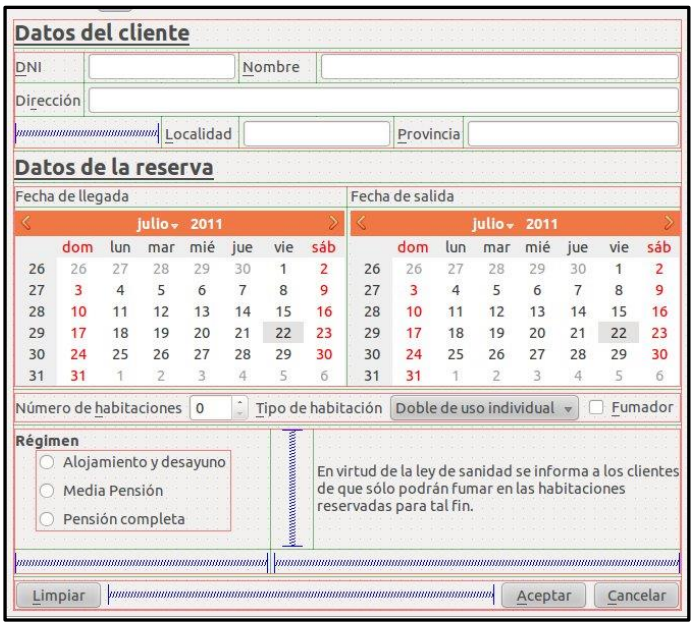


Ahora guardamos la interfaz en un directorio que tengamos preparado para ello, por ejemplo, en nuestro caso hemos creado el directorio **ProyectoHotel** dentro de **misInterfaces** De esta forma tendremos un directorio para cada proyecto que hagamos con QT Designer. Para guardar una interfaz basta con hacer clic en **File >> Save as...**

Anexo II - Añadir funcionalidad a la interfaz

Descripción de la interfaz

En puntos anteriores hemos generamos esta interfaz para gestionar, en un modo muy simple, las reservas de un hotel:



Creamos un cuadro de diálogo y añadimos los elementos gráficos de interfaz, hemos modificado algunas características básicas como los textos asociados que aparecen en la interfaz, los nombres de los objetos, y otras propiedades necesarias para completar su funcionamiento. También se han asociado las etiquetas a sus objetos para permitir los atajos de teclado y se ha establecido el orden de tabulación. Se han ordenado y distribuido adecuadamente los objetos de esta interfaz mediante layouts.

Ahora vamos a hacer que, al producirse determinado acciones sobre los elementos de la interfaz se realicen las tareas que se indiquen.

Funcionalidad que se pretende añadir

- Vamos a empezar con algo sencillo, los botones **OK** y **Cancel**, cierran ambos el diálogo, el valor devuelto depende de cómo se cierre.

- El botón Limpiar hacer que el resto de los elementos de la interfaz vuelvan a sus valores originales.

Para asignar estas acciones a los botones, tenemos que poner el editor en modo "**Edit Signal/Slot**":

Puesto que los botones **OK** y **Cancel** habían sido incluidos por defecto, también por defecto llevan asociada su funcionalidad.

Para el botón Limpiar haremos lo siguiente:

1. Hacemos clic con el ratón sobre el botón y sin soltarlo lo arrastramos hasta el primer cuadro de texto: DNI.
2. Al soltar el ratón aparecerá una ventana como la siguiente: A la izquierda aparece el objeto sobre el que se emite la señal y a la derecha el objeto sobre el que recae la acción (también puede ser el dialogo). En nuestro caso el emisor es un pushButton, con la lista de señales que se ve en la imagen y el receptor un cuadro de texto, con las acciones que puede ejecutar. Cuando pulsemos el botón (**clicked**) el campo de texto se borrará (**clear()**). Si no encontramos la función que nos haga falta pulsamos en Show signals and slots inherit from QWidget para ver más señales y ranuras.
3. Repetimos el proceso para todos los elementos que pueden volver a su estado inicial de la interfaz: campos de texto, spinbox, lista desplegable, casilla de verificación y botones de radio. En todos los casos elegimos la señal clicked y la función clear().

,

ÍNDICE