Capítulo 4: XML Schema.

4. XML Schema.

En este capítulo vamos a ver en qué consiste XML Schema. Ya introdujimos algunas de sus características en el capítulo anterior, y ahora profundizaremos en todas ellas. Comenzaremos por definir qué es un Esquema XML, viendo después las ventajas que presenta frente a una DTD. A continuación realizaremos una introducción a conceptos fundamentales en esta tecnología. Seguiremos un ejemplo guía que nos facilite la comprensión de todos estos conceptos. En los últimos apartados explicaremos los tipos simples y complejos en XML Schema: es ahí donde describiremos todas las reglas y la sintaxis de este tipo de esquema.

4.1-.Definición.

Un esquema, al igual que una DTD, define la "apariencia" de uno o más documentos XML; esto es, qué elementos y atributos pueden contener, en qué orden deben estar, y cuál puede ser su contenido. De hecho, podríamos calificar las DTD como un tipo de esquema, en el sentido amplio de la palabra.

Un esquema puede ser visto como una colección (vocabulario) de definiciones de tipos y declaraciones de elementos cuyos nombres pertenecen a un determinado espacio de nombres llamado espacio de nombres de destino. Los espacios de nombres de destino hacen posible la distinción entre definiciones y declaraciones de diferentes vocabularios.

En un principio existían dos sistemas fundamentales para describir la apariencia de los documentos XML: las DTD y XML Schema (se utiliza el término "Esquema" para referirnos a este último). Sin embargo, han ido apareciendo nuevos sistemas como pueden ser RELAX-NG o Schematron para realizar esta descripción de la apariencia de los documentos XML. Nos centraremos en este capítulo en la explicación de los Esquemas XML, dado que la Herramienta desarrollada en este proyecto está enfocada a la manipulación de este tipo de documentos.

4.2-. Ventajas de XML Schema sobre las DTD.

Antes de entrar en detalle, vamos a resaltar las principales ventajas de XML Esquema sobre la solución tradicional de las DTDs en lo que respecta a la descripción de los documentos XML. Un esquema escrito con XML Esquema es similar a una DTD, en la medida que se usa para establecer el esquema o estructura de una clase de documentos XML. Al igual que ocurre con las DTD, el papel de los Esquemas XML consiste en describir los elementos y sus modelos de contenido, de forma que los documentos puedan ser validados. No obstante, XML Esquemas irá mucho más allá que las DTD, permitiendo asociar tipos de datos con elementos y también con atributos.

La primera ventaja que encontramos es que en una DTD, el contenido de los elementos se limita a cadenas y a otros tipos de datos primitivos. XML Schema soporta una amplia gama de tipos de datos, como por ejemplo enteros, números de coma

flotante, fechas y horas. XML Schema también incluye soporte para otras características, como el modelo de contenido abierto y la integración para espacios de nombres. Aparte de ofrecer estas características, XML Schema es completamente distinto (visualmente hablando) a las DTD, ya que se basa en sintaxis XML para escribir los documentos de esquemas, cosa que no hacen las DTD; digamos que un Schema XML que define la "apariencia" de un documento XML está escrito con el propio lenguaje XML. Se trata de una mejora importante en comparación con las DTD, ya que no implica tener que aprender un lenguaje diferente. Sólo habrá que a aprender a usar el vocabulario XML Schema.

A continuación se listan las principales ventajas que ofrece XML Schema con respecto a las DTD:

- XML Schema se basa en XML y no en una sintaxis especializada.
- XML Schema puede ser analizado sintácticamente y manipulado como cualquier otro documento XML.
- XML Schema soporta la integración de los espacios de nombre, lo cual le permite asociar nodos individuales de un documento con las declaraciones de tipo de un esquema.
- XML Schema soporta grupos de atributos, lo que le permite lógicamente combinar atributos.

A pesar de todas estas ventajas, en un principio estaba más difundido el uso de las DTD, seguramente por costumbre y facilidad. Debido a las grandes ventajas comentadas anteriormente, el uso de XML Schema ha ido avanzando rápidamente, llegando incluso a reemplazar a las DTD.

4.3-. Conceptos Fundamentales.

4.3.1-. Tipos Simples y Complejos.

En un esquema podemos dividir un documento en dos tipos de contenidos: simples y complejos. Los elementos del tipo simple son aquellos que sólo pueden contener texto. Los del tipo complejo pueden contener otros elementos, incluso atributos. Los atributos son considerados del grupo simple, ya que contiene solamente texto.

Los elementos del tipo complejo tienden a describir la estructura de un documento más que su contenido. Hay cuatro clases fundamentales de tipos complejos:

- Elementos "sólo elementos", que contienen otros elementos o atributos, pero no contienen texto.
- Elementos "vacíos", que posiblemente contengan atributos, pero nunca elementos ni texto.
- Elementos de "contenido mixto", que contienen una combinación de elementos, atributos y/o texto, especialmente elementos y texto
- Elementos de "solo texto", que contienen sólo texto y atributos.

Todos pueden contener atributos. Es más, todo elemento que contenga un atributo será de tipo complejo.

Tanto los tipos simples como los personalizados pueden tener nombre, en cuyo caso podrán utilizarse en otras partes del esquema, o bien ser anónimo, un cuyo caso sólo se utilizan en el elemento en el que aparece la definición.

4.3.2-. Declaraciones locales y globales.

En XML Schema, el contexto es muy importante. Los componentes del esquema, bien sean elementos, atributos, tipos simples o complejos, grupos o grupos de atributos, que son declarados en el nivel superior del esquema (debajo del elemento xsd:Schema*)*, son considerados declarados globalmente, y están disponibles para ser utilizados en el resto del esquema. Sin embargo, las declaraciones globales de elementos no determinan el lugar donde puede aparecer un elemento en el documento XML, sólo su apariencia. Por tanto, debemos hacer referencia a la declaración global de un elemento para que realmente aparezca en el documento XML.

La única excepción a esta regla está destinada para el elemento raíz, el cual es referenciado automáticamente, independientemente de si se ha declarado globalmente o no.

Cuando se define un tipo complejo, éste puede hacer referencia a los elementos existentes declarados globalmente, o bien declarar y definir nuevos elementos. Estos elementos declarados localmente están limitados a la definición de tipo complejo en el cual han sido declarados y puede que no se utilicen en ninguna otra parte del esquema. Además, sus nombres sólo necesitan ser únicos en el contexto en el que aparecen. Estos elementos son referenciados automáticamente, es decir, la posición en la que son definidos también determina en qué parte del documento XML deben aparecer.

4.3.3-. Conflictos de Nombres.

En XML Schema se puede utilizar el mismo nombre para cosas diferentes, pero hay que tener cuidado con esto. ¿Qué ocurre si le damos el mismo nombre a dos cosas diferentes? La respuesta depende de las dos cosas en cuestión, aunque en general, cuanto más parecidas son ambas cosas, más probabilidades de que exista un conflicto.

He aquí algunos ejemplos para ilustrar cuando los nombres iguales causan problemas:

- Si las dos cosas son tipos: supongamos que definimos un tipo complejo llamado EstadoEEUU y un tipo simple llamado EstadoEEUU, existirá conflicto.

- Si las dos cosas son un tipo y un elemento o atributo: imaginemos que definimos un tipo complejo llamado DireccionEEUU y que declaramos un elemento llamado DireccionEEUU, entonces no existirá conflicto.

- Si las dos cosas son elementos dentro de tipos diferentes, (es decir, no son elementos globales): por ejemplo, declaramos un elemento llamado nombre

como parte del tipo DireccionEEUU y un segundo elemento llamado nombre como parte del tipo Elementos, no existe conflicto. (Tales elementos son a menudo denominados declaraciones de elementos locales).

- Finalmente, si las dos cosas son tipos y tú defines una y el Esquema XML ha definido la otra: por ejemplo, supongamos que tú has definido un tipo simple llamado decimal, que es algo que ya existe en la sintaxis XML Schema como veremos un poco más adelante, no existe conflicto. La razón de esta contradicción aparente en este último ejemplo es que los dos tipos pertenecen a espacios de nombres distintos.

4.3.4-. Comienzo de un esquema simple.

Un esquema es un documento XML que sólo contiene texto y aparece con la extensión .xsd. Comienza con una declaración XML estándar, seguida de una declaración del espacio de nombre de XML esquema. Por ejemplo:

```
<?xml version=”1.0 “?>
<xsd:schema xmnls:xsd= http://www.w3.org/2001/XMLSchema>
        ( aquí irán el conjunto de reglas del esquema)
</xsd:schema>
```

Para comenzar un esquema, los pasos a realizar son:

- Escriba la declaración XML, similar a *<?xml version=”1.0 “?>*, al comienzo del documento.
- Escriba <xsd:schema.
- Escriba xmnls:xsd= “http://www.w3.org/2001/XMLSchema” para declarar el espacio de nombre del “esquema de esquemas”. A partir de ahora cualquier documento o escritura que aparezca con el prefijo xsd: delante será reconocido como si procediera de este espacio de nombre.
- Escriba > para completar el elemento de apertura.
- Tras la definición del propio esquema se completa el documento con </xsd:schema>.
- Se guarda el esquema como archivo de sólo texto y con extensión .xsd.

4.4-. Ejemplo Guía.

Para continuar con la explicación, y para que resulte más sencillo, vamos a ir viendo un ejemplo. Consideremos un documento XML en un fichero llamado po.xml. Describe una hoja de pedido generada por un pedido de productos para el hogar y una aplicación de facturación:

-- Hoja de Pedido, po.xml

```
<?xml version="1.0"?>
<hojaPedido fechaPedido="1999-10-20">
        <enviarA pais="EEUU">
                <nombre>Alice Smith</nombre>
                <calle>123 Maple Street</calle>
                <ciudad>Mill Valley</ciudad>
                <estado>CA</estado>
                <zip>90952</zip>
        </enviarA>
        <facturarA pais="EEUU">
                <nombre>Robert Smith</nombre>
                <calle>8 Oak Avenue</calle>
                <ciudad>Old Town</ciudad>
                <estado>PA</estado>
                <zip>95819</zip>
        </facturarA>
        <comentario>¡Deprisa, mi césped parece una selva! </comentario>
        <elementos>
            <elemento numProducto="872-AA">
                <nombreProducto>Cortacesped</nombreProducto>
                <cantidad>1</cantidad>
                <precioEEUU>148.95</precioEEUU>
                <comentario>Confirmar que es eléctrico</comentario>
            </elemento>
            <elemento numProducto="926-AA">
                  <nombreProducto>Monitor para bebes </nombreProducto>
                  <cantidad>1</cantidad>
                  <precioEEUU>39.98</precioEEUU>
                  <fechaEnvio>1999-05-21</fechaEnvio>
            </elemento>
        </elementos>
</hojaPedido>
```

La hoja de pedido consiste de un elemento principal, hojaPedido, y los subelementos enviarA, facturarA, comentario, y elementos. Estos subelementos (excepto comentario) además contienen otros subelementos, y así, hasta que un subelemento como precioEEUU contiene un número en vez de más subelementos. Los elementos que contienen subelementos o tienen atributos son denominados tipos complejos, mientras que los elementos que contienen números (o cadenas de caracteres, o fechas, etc.) pero no contienen subelementos ni atributos se denominan tipos simples. Algunos elementos tienen atributos; los atributos siempre son tipos simples.

Los tipos complejos en el documento instancia, y algunos de los tipos simples, están definidos en el esquema para hojas de pedido. Los demás tipos simples están definidos como parte del repertorio de tipos simples predefinidos del Esquema XML.

El esquema de la hoja de pedido está contenido en el archivo po.xsd:

-- El Esquema de la Hoja de Pedido , po.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation xml:lang="es">
                Esquema de hoja de pedido para Example.com.
                Copyright 2000 Example.com. Todos los derechos reservados.
        </xsd:documentation>
    </xsd:annotation>

  <xsd:element name="hojaPedido" type="TipoHojaPedido"/>

 <xsd:element name="comentario" type="xsd:string"/>

 <xsd:complexType name="TipoHojaPedido">
        <xsd:sequence>
                <xsd:element name="enviarA" type="direccionEEUU"/>
                <xsd:element name="facturarA" type="direccionEEUU"/>
                <xsd:element ref="comentario" minOccurs="0"/>
                <xsd:element name="elementos" type="Elementos"/>
        </xsd:sequence>
        <xsd:attribute name="fechaPedido" type="xsd:date"/>
 </xsd:complexType>

 <xsd:complexType name="direccionEEUU">
        <xsd:sequence>
                <xsd:element name="nombre" type="xsd:string"/>
                <xsd:element name="calle" type="xsd:string"/>
                <xsd:element name="ciudad" type="xsd:string"/>
                <xsd:element name="estado" type="xsd:string"/>
                <xsd:element name="zip" type="xsd:decimal"/>
        </xsd:sequence>
        <xsd:attribute name="pais" type="xsd:NMTOKEN" fixed="EEUU"/>
 </xsd:complexType>

 <xsd:complexType name="Elementos">
  <xsd:sequence>
     <xsd:element name="elemento" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element name="nombreProducto" type="xsd:string"/>
                        <xsd:element name="cantidad">
                                <xsd:simpleType>
                                        <xsd:restriction base="xsd:positiveInteger">
                                                <xsd:maxExclusive value="100"/>
                                        </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:element>
                        <xsd:element name="precioEEUU" type="xsd:decimal"/>
```

```
                    <xsd:element ref="comentario" minOccurs="0"/>
                    <xsd:element name="fechaEnvio" type="xsd:date"
                    minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="numProducto" type="SKU" use="required"/>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
</xsd:complexType>

  <!-- Stock Keeping Unit [Código de Almacenaje], -->
  <!-- un código para identificar productos -->

  <xsd:simpleType name="SKU">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="\d{3}-[A-Z]{2}"/>
        </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

El esquema consiste de un elemento schema y una variedad de subelementos, los más notables, element (elemento), complexType (tipo complejo), y simpleType (tipo simple) que determinan la aparición de elementos y su contenido en los documentos instancia (los documentos XML).

Cada uno de los elementos del esquema tiene el prefijo “xsd:” que está asociado con el espacio de nombres de XML Schema. El prefijo xsd: es usado por convención para denotar el espacio de nombres del XML Schema, aunque puede utilizarse cualquier prefijo. El mismo prefijo, y por tanto la misma asociación, también aparece en los nombres de los tipos simples predefinidos, por ejemplo xsd:string. El propósito de la asociación es identificar los elementos y tipos simples como pertenecientes al vocabulario del lenguaje XML Schema y no al vocabulario del autor del esquema.

4.5-. Tipos Simples.

Todos los elementos de tipo simple se parecen en que sólo contienen texto y no tienen atributos. No hay un único elemento de tipo simple, hay muchos; e incluso puede crearse uno propio. Por ejemplo, *string* es un tipo simple que permite al elemento contener cualquier clase de texto, (similar al #PCDATA en las DTD).

Por ejemplo, supongamos que queremos usar un elemento precio en nuestro documento XML. Necesitaríamos algo así:

<precioUSDollars>24.50</precioUSDollars>

Sería fácil ajustarlo para que pudiera contener sólo números. Esta es una forma:

<xsd:element name=”precioUSDollars” type=”xsd:decimal” />

El tipo decimal restringe el texto en el interior del elemento precioUSDollars para que contenga sólo números (positivos o negativos). Cualquier otra cosa producirá un documento XML no válido.

Una primera clasificación de los tipos simples puede hacerse en:

- Tipos simples predefinidos por XML Esquema, también conocidos como tipos primitivos o atómicos (nombres usados en la norma del W3C).

- Tipos simples personalizados.

4.5.1-. Tipos Simples Predefinidos.

Veamos los tipos simples predefinidos, que a su vez se dividen en:

- Tipos simples basados en números:

xsd:decimal

Un elemento con tipo simple decimal puede ser cualquier número positivo o negativo.

xsd:integer

Un elemento con tipo simple integer puede ser cualquier número entero positivo, negativo o cero.

xsd:positiveInteger

Un elemento con el tipo simple positiveInteger puede ser cualquier número entero positivo, pero no puede ser negativo ni cero.

xsd:negativeInteger

Un elemento con el tipo simple negativeInteger puede ser cualquier número entero negativo pero ni positivo ni cero.

xsd:nonPositiveInteger

Un elemento con el tipo simple nonPositiveInteger puede ser cualquier número negativo o cero, pero no positivo.

xsd:nonNegativeInteger

Un elemento con el tipo simple nonNegativeInteger puede ser cualquier número positivo o cero, pero no negativo.

xsd:float

Un elemento con el tipo simple float puede ser un número de coma flotante de 32 bits, como 3.5 ó 4.5e+08. También puede ser infinito positivo (INF), infinito negativo (-INF), y “no es un número” (NaN).

xsd:double

Un elemento con el tipo simple double puede ser un número de coma flotante de 64 bits, con el mismo formato que float.

- Tipos simples basados en fechas y horas.

Podemos especificar que un elemento contenga solamente información de fecha y hora. El formato de esta información es predeterminado. Por ejemplo, las fechas están en formato AAAA - mm - dd, de modo que para la fecha “28 de Octubre de 2003” escribiremos 2003 - 10 - 28.

A continuación detallaremos los tipos simples dedicados a fecha y hora:

xsd:date

Un elemento date debe tener el formato AAAA – mm – dd .

xsd:time

Un elemento time debe tener el formato hh:mm:ss.sss. Se puede añadir también una Z opcional, y + hh:mm ó – hh:mm al final para indicar una zona horaria diferente. Usamos Z si la hora se ajusta a la Hora Media de Greenwich (GMT – Greenwich Mean Time) o la Hora Universal Coordinada (UTC – Coordinate Universal Time), o utilizaremos las horas y minutos adicionales para indicar diferencias respecto al GMT. Las horas se datan según el sistema de 24 horas.

xsd:timeInstant

Este tipo permite combinar data y time en una sola cadena. Los formatos son los descritos, excepto por una “T” entre fecha y hora: AAAA – mm – dd T hh:mm:ss.sss, incrementando si es preciso con la información adicional de zona horaria.

xsd:timeDuration

Este tipo es un poco raro. El formato es: PnYnMnDnTHnMnSn. “P” es igual al periodo y es de uso obligatorio. Las otras letras estarán presentes solo si se van a usar, excepto la “T” que también debe aparecer.

xsd:month

Este tipo especifica un mes y año. El formato es AAAA – MM.

xsd:year

Este tipo especifica un año. Su formato es AAAA.

xsd:century

Este tipo especifica un siglo; se marca como los dos primeros dígitos de los cuatro de un año, incrementados en uno. No hace falta recordarlo pero a veces la inercia nos confunde.

xsd:recurringDate

Este tipo especifica un mes y un día, pero no el año. El formato es el siguiente: -- MM – DD. Observe el guión doble al principio.

xsd:recurringDay

Este tipo especifica un día en el mes. El formato es: --- DD.

- Tipos simples variados:

xsd:boolean

Este tipo especifica “verdadero” (true) o “falso” (false); también puede usarse el 0 y el 1.

xsd:language

Este tipo especifica un idioma, en el formato de dos caracteres establecido por la norma ISO639. Por ejemplo, el inglés es “EN”, y el español es “ES”.

xsd:uri-reference

Aunque se escriba así, este tipo no especifica – todavía – un URI (Universal Resource Identifier, identificador universal de recursos), sino un URL (Uniform Resource Locator, localizador uniforme de recursos).

xsd:NMTOKEN

Este tipo obliga al texto a ser un nombre XML válido

Esto es todo lo que hay respecto a los tipos simples básicos. Realmente es mucho más de lo que puede ofrecer una DTD. Para el esquema XML, en cambio, apenas es la plataforma de lanzamiento. Usando estos tipos simples y unas pocas restricciones podemos crear nuestros propios tipos.

4.5.2-. Tipos Simples Personalizados.

Estos tipos de datos los usamos cuando no queremos amoldarnos a ninguno de los anteriores, donde tenemos unas reglas exactas. Para ello lo que hacemos es coger un tipo simple predefinido como base y, a partir de él, hacemos las restricciones oportunas para modelarlo a nuestro problema.

La definición de atributos en una DTD nos permite especificar una lista de valores aceptables. Es mejor que nada, pero todavía sigue siendo limitado. El Esquema XML nos proporciona un medio para especificar texto aceptable en un elemento de tipo simple. Para ello vamos a hacer uso del elemento restriction. El formato de este elemento es: <xsd:restriction base=“base”>, donde base es el tipo simple sobre el que queremos imponer la restricción. A continuación se especifican las características particulares del tipo (utilizando las propiedades que veremos a continuación). Para finalizar la restricción utilizamos la correspondiente etiqueta de cierre </xsd:restriction>.

Veamos distintos tipos de restricciones que se pueden realizar.

- Forzar texto para que se adapte a un patrón:

En algunas ocasiones se necesita que los datos se ajusten a un patrón determinado, como un código postal, una fecha larga o cualquier otro tipo. Para crear ese patrón, puede usarse un lenguaje especial llamado "expresión regular" que establece dónde pueden aparecer un tipo determinado de caracteres.

Uso de un patrón:

```
<xsd:element name="phone_number">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="(\d{3})\s\d{3} - \d{4}" />
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
```

El nuevo elemento usado es xsd:pattern. Este elemento de esquema sólo atribuye su valor (value), el cual contiene la declaración de expresión regular. Algunos de los parámetros más útiles son:

\d: *cualquier dígito.*
\D: *cualquier no-dígito.*
\s: *espacio en blanco, retorno de carro, línea nueva, o intro.*
\S: *cualquier carácter distinto a espacio en blanco.*
(ab)*: *cualquier cosa entre paréntesis puede aparecer cero o más veces.*
(ab)?: *cualquier cosa entre paréntesis puede aparecer una o más veces.*
a{n}*: "a" puede aparecer en una cadena n veces.*

Así, la expresión regular que hemos usado en el ejemplo anterior significa, describiéndolo con palabras:

- Debe empezar con paréntesis de apertura '('.
- Seguido por tres dígitos.
- Seguidos por un paréntesis de cierre ')' y un espacio.
- Seguidos por tres dígitos más.
- Seguidos por un guión.
- Y terminar con cuatro dígitos.

- Limitación de valores numéricos:

Hay cuatro medios para limitar los valores numéricos de un tipo: maxInclusive, minInlclusive, maxExclusive, minExclusive.

Imposición de valores máximos y mínimos:

```
<xsd:element name="numShortsDiscountOrder">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
```

```
        <xsd:maxInclusive value="2000" />
        <xsd:minInclusive value ="500" />
      </xsd:restriction>
    </xsd:simpleType>
  <xsd:element>
```

Obsérvese que ahora partimos para nuestro tipo simple como xsd:integer. Para ajustar un límite superior, debe usarse xsd:maxInlusive, que en este caso es 2000. Esto significa que el texto dentro de numShortsDiscountOrder debe ser un número menor o igual que 2000. Si hubiéramos usado xsd:maxExclusive, el texto debería haber sido un número menor, pero no igual a 2000. Para establecer un límite inferior hemos usado xsd:minInlusive, de modo que el número en el elemento numShortsDiscountOrder debe ser mayor o igual que 500.

Existe algún método más para constreñir valores numéricos en XML. Esos métodos implican limitar la cantidad de dígitos en el número; podemos limitar la cantidad total de dígitos en un número y la cantidad máxima de dígitos a la derecha del punto decimal. Por ejemplo, para forzar la precisión en los números podríamos realizar lo siguiente:

```
<xsd:element name="scienceNum">
  <xsd:simpleType>
    <xsd:restriction base="xsd:decimal">
      <xsd:precision value="6" />
      <xsd:scale value="3" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Aquí nos encontramos dos nuevos elementos de XML Schema: xsd:precision y xsd:scale. xsd:precision coloca un límite superior sobre la cantidad total de dígitos que puede tener un número, mientras que xsd:scale controla el máximo de dígitos a la derecha del punto decimal. Las dos líneas que siguen serían válidas:

```
<scienceNum>123.456</scienceNum>
<scienceNum>7</scienceNum>
```

- Limitación de la longitud de las cadenas:

Imaginemos que estamos anotando el pedido de un comprador, y que una parte del mismo corresponde al estado o provincia donde vive nuestro cliente. Queremos limitar ese campo a una abreviatura de dos caracteres. Para hacerlo, podemos usar un elemento de XML Schema denominado xsd:lenght.

```
<xsd:element name="state">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:lenght value="2" />
    </xsd:restriction>
```

```
  </xsd:simpleType>
</xsd:element>
```

Este ejemplo es muy sencillo: permite sólo dos caracteres en el elemento state.

También podemos establecer límites superiores e inferiores en una cadena de texto. Por ejemplo, si queremos permitir tanto abreviaturas como nombres completos. Esto significaría que queremos un mínimo de dos y un máximo de, digamos, trece caracteres.

```
<xsd:element name="state">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:minLenght value="2" />
      <xsd:maxLenght value="13" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Vemos dos nuevos elementos de Esquema XML: xsd:minLenght y maxLenght. No necesitan demasiada explicación; ajustan el mínimo y el máximo para la longitud de una cadena de texto.

- Creación de listas:

Todos los tipos simples vistos hasta ahora contienen solo un ítem por elemento: es decir, un único número, fecha o cadena. Si quisiéramos, por alguna razón que nuestro elemento contuviera una lista de ítems separados por un espacio en blanco, XML Schema tiene preparado el tipo para hacerlo: xsd:list. Veamos la creación de una lista:

```
<xsd:element name="sizes">
  <xsd:simpleType>
    <xsd:list itemType="xsd:string" />
  </xsd:simpleType>
</xsd:element>
```

Obsérvese que, en vez de xsd:restriction, se ha usado xsd:list, que permite crear una lista de ítems separados por un espacio en blanco. He aquí un ejemplo que sería válido:

```
<sizes>XL XXL S XS M</sizes>
```

También puede limitarse el tamaño de la lista agregando algunos parámetros en su interior:

```
<xsd:list itemType="xsd:string">
        <xsd:lenght value="5" />
</xsd:list>
```

Este código limitará el tamaño de la lista a cinco ítems, ni uno más ni uno menos. Pero podríamos haber usado:

```
<xsd:list itemType=“xsd:string”>
        <xsd:minLenght value=“2” />
        <xsd:maxLenght value=“7” />
</xsd:list>
```

En esta ocasión, la lista con espacio delimitado puede contener cualquier entidad entre dos y siete ítems. Otras limitaciones que pueden usarse son xsd:pattern, xsd:enumeration (véase su uso en el ejemplo del siguiente apartado) y xsd:whitespace.

- Combinación de tipos simples:

No solo tiene una impresionante libertad para determinar el tipo de datos de su XML; también puede combinar varios tipos simples en un tipo único y más grande. Por ejemplo, habría dos modos de describir el momento en el que un cliente hace un pedido a nuestra tienda: mediante la fecha actual, o mediante los valores "hoy", "mañana" o "ayer" (no es muy recomendable hacer esto último, aunque sí es posible). Para combinar esos dos tipos, introduciremos un nuevo elemento de Esquema XML: xsd:union.

```
<xsd:element name=“orderDate”>
<xsd:simpleType>
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base=“xsd:string”>
        <xsd:enumeration value=“yesterday” />
        <xsd:enumeration value=“today” />
        <xsd:enumeration value=“tomorrow” />
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
       <xsd:restriction base=“xsd:date” />
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
</xsd:element>
```

Es un poco árido leer todos los elementos de esquema anidados, pero la estructura general es muy sencilla. Dentro del elemento xsd:union listamos todos los tipos simples que queremos usar, uno tras otro. Para este ejemplo, he aquí un XML válido:

```
<orderDate>yesterday</orderDate>
<orderDate>2003-10-28 </orderDate>
```

Ambas líneas son XML válidas, porque contiene o bien una fecha o bien una de las cadenas permitidas.

- Determinar el contenido de un elemento:

También se puede ejercer un control total sobre el contenido de un elemento y ajustarlo explícitamente, o puede simplemente darle un valor predeterminado. Véase en el ejemplo:

```
<xsd:element name="color" type="xsd:string" fixed="blue" />
<xsd:element name="size" type="xsd:string" default="M" />
```

Para el elemento color, el único XML posible es <color>blue</color>. No hay otra opción. Para la talla, en cambio, es posible cualquier cadena y, si se omite el elemento en XML, el analizador colocará automáticamente <size>M</size> en el hueco adecuado.

- Tipos simples personales reutilizables:

En ocasiones puede ser útil crear un tipo simple a medida que pueda ser usado por más de un elemento. Por ejemplo, podría ser que necesitáramos varios números de teléfono procedentes de un cliente empresarial: principal, móvil y fax. Sería muy molesto tener que repetir los elementos <xsd:simpleType> para cada entrada. Afortunadamente, hay un medio simple para definir un tipo simple y reutilizarlo una y otra vez. Todo lo que se tiene que hacer es crear un tipo simple que no esté dentro de un xsd:element y darle un nombre. Por ejemplo:

```
<xsd:simpleType name="numTelefono">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value=" (\d{3})\s\d{3} - \d{4}" />
  </xsd:restriction>
</xsd:simpleType>

</xsd:element name="principal" type="numTelefono"/>
</xsd:element name="movil" type="numTelefono"/>
</xsd:element name="fax" type="numTelefono"/>
```

Este código es muy similar al visto antes, excepto en que xsd:simpleType no se introduce en xsd:element y tiene su propio nombre. Ese nombre podrá ser reutilizado, ajustando el tipo de otros elementos en su esquema XML.

4.6-. Tipos Complejos.

Un elemento que puede contener otros elementos o al que se le permite contener atributos, se considera que tiene un tipo complejo. Debido a que muchos documentos XML tienen elementos que contienen otros elementos, los tipos complejos son muy usados dentro de la definición de un esquema.

Como ya vimos cuando se presentaron los tipos complejos, existen cuatro clases de tipos complejos en función de su contenido:

- Elementos “sólo elementos”, que contienen otros elementos o atributos, pero no contienen texto.
- Elementos “vacíos”, que posiblemente contengan atributos, pero nunca elementos ni texto.
- Elementos de “contenido mixto”, que contienen una combinación de elementos, atributos y/o texto, especialmente elementos y texto
- Elementos de “solo texto”, que contienen sólo texto y atributos.

Las definiciones de cualquier tipo complejo van a tener en común entre si el comenzar la definición con xsd:complexType, a partir de ahí detallaremos las características del tipo según la clasificación anterior.

- Definición de un tipo que contienen sólo elementos.

Escribiremos <*xsd:complexType name= “etiqueta”*> para comenzar la definición del tipo donde etiqueta indica el nombre de referencia del tipo en cuestión. A continuación se presenta la estructura y características del tipo complejo. Para ello disponemos de una serie de etiquetas que nos proporcionan una serie de opciones. Pasamos a verlas en detalle.

a) Cómo solicitar elementos para aparecer en secuencia:

Para ello hacemos uso de opción <*xsd:sequence*>. Esto significa que los elementos que se detallen a continuación deben aparecer todos y en el orden indicado. Los elementos introducidos pueden ser a su vez simples o complejos, para estructuras anidadas. No hay que olvidar que tras la declaración de elementos debemos cerrar con la correspondiente etiqueta </*xsd:sequence*>.

Si queremos definir la incardación (es decir, el número de apariciones posibles) de un elemento en un Esquema XML, hacemos uso de los atributos minOccurs y maxOccurs dentro de la etiqueta de cada elemento introducido en la secuencia. Podemos usar tanto uno como otro, no tienen por qué estar ambos si no queremos, (si bien *maxOccurs* debe ser igual o mayor que *minOccurs*). Estos dos atributos tienen valores predeterminados: ambos estarán ajustados a 1 a no ser que se indique otra cosa.

Veamos la equivalencia de estos atributos con la DTD:

XML Schema	DTD equivalente
minOccurs=”0” maxOccurs=”unbounded”	element*
MinOccurs=”0” maxOccurs=”1”	element?
minOccurs=”1” maxOccurs=”unbounded” (sin límites)	element+
minOccurs=”1” maxOccurs=”1”	element

Tabla 3: Equivalencias entre atributos XML Schema y DTD.

Los atributos minOccurs y maxOccurs no se pueden utilizar con elementos declarados globalmente, sólo pueden ir acompañados de elementos declarados localmente y con referencia a los globales. Se pueden usar en *xsd:sequence*, *xsd:choice*, *xsd:all*, y en referencia de grupos de con nombres. Veremos a continuación estas opciones que acabamos de nombrar.

b) Para ofrecer una opción.

Se ha podido observar que *xsd:sequence* es un poco rígido. Con *xsd:choice* obtenemos una opción interesante si se tiene una lista de elementos (o grupos de elementos), y XML debe elegir de esa lista.

La sintaxis para usar *xsd:sequence* y *xsd:choice* es exactamente la misma. Los grupos de elementos que tienen elecciones o secuencia pueden estar anidados unos en otros.

c) Para permitir a los elementos aparecer en cualquier orden.

Si queremos que un elemento pueda contener otros elementos en cualquier orden, podemos agrupar esos elementos internos con un grupo all. Las etiquetas son *<xsd:all>* y *</xsd:all>*. Se comportan igual que xsd:sequence, excepto en que los elementos no tienen que aparecer en ningún orden predeterminado.

- Atributos.

Antes de pasar a ver el resto de tipos complejos conviene pararnos en los conceptos relacionados con los atributos, dado que los tipos a ver los van a poder incluir.

a) Definición de atributos.

Un atributo es siempre de tipo simple: no contiene ningún otro elemento o atributo, pudiendo ser también un tipo simple hecho a medida, pero no un tipo complejo. Siempre aparece en un elemento de tipo complejo, es decir, sus declaraciones están anidadas en el interior de *xsd:complexType* (salvo, claro está, las declaraciones globales).

Para declarar un atributo usamos la etiqueta *<xsd:atrribute>*. Este elemento xsd:atrribute debe contener uno de los siguientes atributos:

- type= “tipo”, donde tipo es el tipo al que pertenece el atributo.
- ref= “etiqueta”, donde etiqueta identifica la definición de un atributo que ya se ha declarado (globalmente).

Se pueden añadir facetas de restricción si se desea igual que se hacía en la declaración de un tipo simple. Terminamos la declaración del atributo con la correspondiente etiqueta </xsd:atrribute>.

Además de los dos atributos vistos, xsd:attribute puede tener otros dos más que pasamos a ver a continuación.

b) Usos de un atributo.

Cuando se define un atributo, puede decidirse al mismo tiempo cómo va a ser utilizado. Si va a ser requerido u opcional, y si va a tener un valor predefinido. Los atributos pueden aparecer una vez o ninguna, pero ningún otro número de veces, por eso la sintaxis para especificar ocurrencias de atributos es diferente que la sintaxis para elementos.

Tales posibilidades se determinan con los atributos *use* y *value* en el elemento *xsd:attribute.*

Ítem	**Descripción**
use=”required”	El atributo debe estar presente.
use=”optional”	El atributo puede estar presente o no. Es el valor por defecto, de modo que no es necesario emplearlo.
use=”prohibited”	Está prohibido que el atributo aparezca en el XML.
value=”valor Obligatorio” use=”fixed”	Si el atributo está presente en el documento XML, debe tener el valor indicado en valorObligatorio. Si no está presente no se ajusta a ningún valor.
value=”valorPredeterminado” use=”default”	Si el atributo no está presente en el documento XML, el analizador tiene la orden de insertarlo y darle el valor predeterminado. Si el atributo está en el documento, su valor sobrescribirá al valor predeterminado.

Tabla 4: Atributos use y value del elemento Attribute.

c) Atributos y elementos.

En secciones posteriores veremos como se declaran los atributos en el resto de tipos complejos a ver (que sólo contengan texto, vacíos, o mixtos, es decir, con elementos y texto). En los tipos que sólo contienen elementos también se pueden incluir atributos. En este caso su declaración ira al final, después de todas las demás declaraciones del tipo. No hay ninguna razón real para ello, excepto que fuerza cierta organización en nuestro código.

- Definición de tipos complejos para elementos que contengan sólo texto.

Si lo que se quiere es un tipo simple (cuyo contenido va a ser una clase específica de texto), pero con atributos, debemos usar este tipo de tipo complejo. Para ello comenzamos con la etiqueta *<xsd:complexType name=“etiqueta”>* como todos los tipos complejos. A partir de ahí añadimos:

- *<xsd:simpleContent>*.
- A continuación podemos añadir el elemento <xsd:restriction base="base"> si se quiere limitar la base del tipo simple con facetas adicionales, o <xsd:extension base="base"> si se quiere expandir su tipo simple. En ambos casos base es la definición del tipo simple en la que se basa el nuevo tipo complejo.
- si hemos elegido *xsd:restriction* declaramos ahora las facetas adicionales.
- declaramos los atributos como hemos visto en el punto anterior, o grupos de atributos (se verán más tarde), que aparecerán en los elementos del tipo complejo
- Escribimos *</xsd:extension>* o *</xsd:rextriction >* si es necesario.
- Escribimos *</xsd:simpleContent>*.
- Por último cerramos el tipo complejo con *</xsd:complexType>*.

- Definición de un tipo complejo para elementos vacíos.

Los elementos que pueden contener atributos pero no tienen nada entre las etiquetas de apertura y cierre se consideran vacíos. Esta vez, en vez de usar la etiqueta xsd:simpleContent usaremos xsd:complexContent. La estructura quedaría:

- Escribimos <xsd:complexType>.
- A continuación <xsd:complexContent>.
- Luego <xsd:extension base="anyType"/> para indicar esencialmente que no hay ningún tipo en el que se pueda basar el tipo complejo. (Y como no tendrá contenido, será mejor).
- Declaramos lo atributos que pueda contener este tipo
- Escribimos </xsd:complexContent> y </xsd:complexType> para completar la declaración.

- Definición de tipos con contenido mixto.

Mientras que el contenido orientado a una base de datos pura rara vez tiene elementos que contengan otros elementos y texto, en los documentos más centrados en el texto no resulta tan extraño. Para indicar esta opción se debe declarar que el contenido va a ser mixto. Esto se hace colocando el atributo del elemento *conplexType*, *mixed* a true. La etiqueta completa quedaría:

<xsd:complexType name="etiqueta" mixed="true">.

A continuación declararemos una secuencia, opción, grupo desordenado o grupo con nombre (que puede contener cualquiera de las tres formas anteriores), para establecer qué elementos podrá contener el tipo complejo. Tras la declaración de elementos vendrá la de atributos o grupos de atributos. Por último </xsd:complexType>.

- Referencia de elementos y atributos.

No es difícil notar que el código de Esquema XML no es tan legible como una DTD. Cuantas más cosas añadamos a un Esquema más difícil resultará su lectura, pues elementos y tipos están anidados los unos en los otros cada vez más profundamente. Un documento medianamente complejo resulta especialmente difícil de leer y casi imposible de depurar. Por suerte, XML Schema proporciona un medio para dividir en módulos el código de forma que sea más fácil de leer, entender y depurar. Este medio consiste en permitir definir un elemento o atributo de forma global y citarlo después.

a) Referencia de elementos:

A los elementos de tipo simple y complejo que se declaren globalmente (dentro del elemento *xsd:schema*), se les debe llamar o hacer referencia en el orden en que aparecen en el documento XML.

Para hacer referencia a un elemento declarado globalmente escribimos *<xsd:element* en cada secuencia, conjunto de opciones, grupo desordenado o definición con nombre en la que aparezca el elemento. A continuación *ref=“etiqueta”*, donde etiqueta es el nombre del elemento declarado globalmente.

Si se desea se puede especificar cuantas veces puede aparecer el elemento en ese punto. Por último añadimos /> para completar la referencia. Un ejemplo de etiqueta sería:

```
<xsd:element name=“nombre” type =“nombreType”/>

<xsd:complexType name=“tipo_animal”>
        <xsd:element ref=“nombre” minOccurs= “2”/>
        <xsd:element name=“amenazas” type=“tipoamenazas”/>
        ….
</xsd:complexType>
```

Se puede hacer mención a un elemento declarado globalmente en tantas ubicaciones como necesitemos.

b) Referencia a atributos:

El mecanismo es el mismo que para los elementos. Una vez que hayamos declarado un atributo o un grupo de atributos se le puede hacer referencia en la declaración de un nuevo atributo mediante el atributo *ref.*

- Tipos complejos nuevos basados en tipos existentes.

Se puede crear tipos complejos a partir de otros tipos complejos ya existentes. El nuevo tipo complejo comienza con toda la información del tipo existente, y se le puede añadir o eliminar prestaciones. Esto puede ser muy útil si se tienen que manejar datos

mediante muchos tipos complejos relacionados entre sí. Éste es un método genial para extender como para dividir en módulos nuestro código

Para declarar este tipo complejo nuevo comenzamos con le etiqueta *<xsd:complexType name =“etiqueta”>* para indicar que se trata de un tipo complejo. El siguiente elemento es *<xsd:complexContent>*.

Para introducir el tipo complejo modelo en el que queremos construir, usamos *xsd:extension* o *xsd:restriction* y llamamos al modelo como la base sobre la cual edificar. Utilizamos *xsd:extension* si estamos construyendo sobre un modelo de tipo complejo (estamos añadiendo cosas, extendiéndolo). Si estuviéramos creando un tipo complejo que contuviera elementos más limitados que aquel en el que nos hemos basado, usaríamos xsd:restriction en lugar de xsd:extension. Escribiríamos:

<xsd:restriction base=“existente”>

O bien:

<xsd:extension base =“existente”>

A continuación declaramos las secuencias adicionales de opciones más o menos restrictivas, o también se puede hacer referencia a los grupos con nombres que debieran formar parte del nuevo grupo. Declaramos o mencionamos los atributos adicionales que formarán parte del nuevo tipo.

Ponemos una etiqueta de cierre acorde con la declaración hecha (xsd:restriction o xsd:extension). Por último: </xsd:complexContent> y </xsd:complexType>.

Veamos un ejemplo completo:

```
<xsd:complexType name=“Tipocaracterísticas”>
        <xsd:sequence>
                <xsd:element name=“peso” type=“xsd:string”/>
                <xsd:element name=“longitud” type=“xsd:string”/>
                <xsd:attribute name=“tipo” type=“xsd:string”/>
        </xsd:sequence>
</xsd:complexType>


/* ----------------------------------------------------------------------------------------*/


<xsd:complexType name=“Tiponacimiento”>
        <xsd:complexContent>
                <xsd:extension base=“Tipocaracterísticas”>
                        <xsd:sequence>
                           <xsd:element name=“madre” type=“xsd:string”/>
                           <xsd:element name=“aniversario” type=“xsd:date”/>
                        </xsd:sequence>
                </xsd:extension>
        </xsd:complexContent>
</xsd:complexType>
```

- Declaración de un elemento de tipo complejo.

Una vez que se ha definido un tipo complejo, le podemos asignar un elemento para que éste pueda usarse en un documento XML. Para declara un elemento de tipo complejo, escribimos <xsd:element para comenzar la declaración del elemento. Luego name="nombre", donde nombre es el nombre de referencia del elemento que estamos definiendo. A continuación el atributo type="etiqueta", donde etiqueta hace referencia a la palabra de identificación que se uso cuando se definió el tipo complejo. Por ultimo cerramos con />.

Hemos de decir que en la declaración de tipos y elementos hay muchas combinaciones posibles. El tipo se puede declarar globalmente y luego incluirse en el elemento, declaración global, o encontrarse definido dentro del propio elemento, declaración local. En este aspecto lo dicho para los tipos simples vale aquí también.

- Elementos con tipos complejos anónimos.

Si no se quiere reutilizar un tipo complejo, puede ser más fácil crear un tipo complejo anónimo en la declaración del elemento.

Para ello declaramos un elemento con <xsd:element name="etiqueta">, donde etiqueta es el nombre del elemento que se declara, o sea, que en el documento XML aparecerá como <etiqueta>. Luego ponemos <xsd:complexType>, en este caso no añadimos el atributo name de complexType. A continuación declaramos una secuencia, opción, grupo desordenado o referencia a un grupo con nombre (que podrá contener cualquiera de las tres formas), para especificar que elementos podrá contener el tipo complejo. Declaramos también o hacemos referencia a los atributos o grupos de atributos que deberán aparecer en los elementos de este tipo si los hay. Escribimos </xsd:complexType> para cerrar la definición del tipo y </xsd:element> para cerrar el elemento.

La única diferencia entre un tipo complejo anónimo y el nombrado, es que con este último se pueden declarar tantos elementos como se quieran, y es, además, la base de los tipos complejos adicionales. El primero sólo se puede usar para definir el elemento en el cual se encuentra.

- Miscelánea.

a) Grupos nombrados de elementos y atributos.

Otro método para controlar la legibilidad y la infraestructura de un documento es la creación de grupos reutilizables de elementos y de atributos. Estos grupos son nombrados al crearlos (es decir, son globales), lo cuál permite referirse a ellos en cualquier otra parte del documento.

- Grupos de elementos:

Si una colección de elementos aparecen juntos en varios lugares del documento XML, se pueden agrupar todos en uno para que sea más fácil referirse a ellos de una sola vez. El grupo de elementos tiene que estar declarado a nivel de la raíz para poder

ser referenciado cualquier cantidad de veces. Comenzamos con la etiqueta *<xsd:group name="etiqueta">*, donde etiqueta es la palabra que identificará a este grupo en particular.

En el interior de dicho grupo, la lista de elementos debe ser sequence, choice o all, incluso se puede añadir otro grupo si así se desea. Lo importante: hay que tener en cuenta que no se puede colocar un atributo dentro de un grupo. Cerramos con *</xsd:group>*.

- Grupos de atributos:

La declaración de un grupo de atributo es equivalente a una de elementos, con la salvedad de que se declaran usando la etiqueta *<xsd:attributeGroup name="label">*.

La referencia a un grupo de atributos o de elementos se hace igual que para elementos y atributos individuales, usando el atributo ref, como en el ejemplo siguiente, donde imágenes y entrenamientos son grupos ya definidos.

<xsd:attributeGroup ref="imagenes">.
<xsd:group ref="entrenamientos">.

b) Anotación y documentación.

Para añadir oficialmente documentación a nuestro Esquema XML y hacerlo más legible y descifrable, incrustamos un elemento xsd:documentation dentro de un xsd:annotation, del siguiente modo:

```
<xsd:annotation>
        <xsd:documentation>
        Escribimos las anotaciones en si.
        </xsd:documentation>
</xsd:annotation>
```

Se pueden colocar estos elementos en cualquier parte de un documento.

c) Incluir archivos externos.

Se puede crear un esquema a partir de distintos documentos utilizando un par de etiquetas. Para incluir un esquema externo completo, debemos emplear el elemento xsd:include, como sigue:

<xsd:include schemaLocation="nombrearchivo.xsd"/>

Usar este elemento de Esquema XML es como copiar y pegar el código desde un archivo externo al nuestro; una inclusión completa. Sin embargo no podremos redefinir nada de lo que se encuentre en el archivo externo.

Existe un rodeo que salva este obstáculo. Si se llama un archivo externo usando *xsd:redefine*, incluiremos el archivo externo y tendremos la oportunidad de redefinir

cualquiera de los elementos, atributos, tipos o grupos de ese archivo. Todas las nuevas definiciones deben ir en el interior del elemento xsd:redefine.

4.7-. Conclusiones. Otros Esquemas.

El gran avance que ha proporcionado XML Schema gracias a todas las características que hemos ido nombrando en esta descripción, nos hace ver la importancia de esta tecnología. Ante una nueva tecnología surge la necesidad de herramientas que permitan su concreción y su uso. Es en este contexto donde surge este proyecto.

Es cierto que la herramienta ha sido implementada en una primera fase para tratar los documentos XML Schema, pero se deja abierta la posibilidad de ampliarlo en un futuro (ver apartado "Conclusiones y Líneas Futuras"), para que no sea una "Herramienta de Edición para XML Schema", sino una "Herramienta de Edición para el Tratamiento de Esquemas". Así, se podría incluir soporte para otros esquemas, como RELAX-NG y Schematron. Se incluyen en la bibliografía algunas referencias que podrían ser útiles para el estudio de dichos esquemas.

A continuación, y dado que el editor ha sido implementado utilizando el lenguaje de programación Java, pasamos a realizar una breve introducción a dicho lenguaje, para después pasar a explicar todo lo relacionado con las interfaces de usuario (GUIs).