Definición de esquemas y vocabularios en xml.

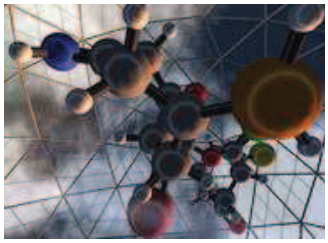
Caso práctico

María había salido a dar un paseo por el puerto, como todos los domingos por la mañana iba **acompañada de su marido José Ramón y su hijo**.
Mientras les veía jugar, **estaba pensando que al día siguiente tenía que hablar con Juan, el técnico** que se encargaba de resolver los problemas de **informática de la empresa de la cual es socia.** Había estado pensando si existiría algún **modo de poder garantizar que la estructura de datos** de cada uno de los **documentos XML que comparte con Félix**, su socio, es la que tiene que ser y no otra, además de asegurar que todos los documentos del mismo tipo mantienen la misma estructura.

Documento XML. Estructura y sintaxis.

Caso práctico

Al día siguiente, **cuando habla con Juan**, sus dudas quedan disipadas. Resulta que hay **varias posibilidades para asegurar una normalización en el formato de los documentos** XML.
Juan comienza por describir la estructura de un documento XML y Félix y María descubren que puede ser un poco más compleja que la que habían estado usando hasta entonces para generar sus documentos.



Hasta ahora hemos trabajado con documentos básicos de XML. Esto significa que dichos documentos están incompletos ya que solo hemos declarado el tipo de documento que va a ser, es decir que ejemplar vamos a definir, pero no hemos definido qué cualidades tiene ese tipo.

En la primera unidad vimos que un documento XML básico estaba formado por un prólogo y un ejemplar.

Recordamos que cada una de esas partes tiene el siguiente cometido:

- **Prólogo:** Informa al intérprete encargado de procesar el documento de todos aquellos datos que necesita para realizar su trabajo. Consta de dos partes:
  - **Definición de XML:** Donde se indica la versión de XML que se utiliza, el código de los datos a procesar y la autonomía del documento. Este último dato hasta ahora siempre ha sido "yes" ya que los documentos generados eran independientes.
  - **Declaración del tipo de documento:** Hasta el momento solo hemos dicho que es el nombre del ejemplar precedido de la cadena <!DOCTYPE y separado de ésta por, al menos un espacio.
    En el apartado siguiente nos encargamos de sus características.
- **Ejemplar:** Contiene los datos del documento que se quiere procesar. **Es el elemento raíz del documento y ha de ser único.** Está compuesto de elementos estructurados según una estructura de árbol en la que el elemento raíz es el ejemplar y las hojas los elementos terminales, es decir, aquellos que no contienen elementos. Los elementos pueden estar a su vez formados por atributos.

Declaración de tipo de documento.

Ya habíamos visto que permite **al autor definir restricciones y características en el documento**, aunque no habíamos profundizado en las partes que la forman:

- **La declaración del tipo de documento propiamente dicha.** Comienza con el texto que indica el nombre del tipo, precedido por la cadena **" "** separado del nombre del tipo por, al menos, un espacio. El nombre del tipo ha de ser idéntico al del ejemplar del documento XML en el que se está trabajando.
- **La definición del tipo de documento.** Permite asociar al documento una definición de tipo DTD , la cual se encarga de definir las cualidades del tipo. Es decir, define los tipos de los elementos, atributos y notaciones que se pueden utilizar en el documento así como las restricciones del

documento, valores por defecto, etc. Para formalizar todo esto, XML está provisto de ciertas estructuras llamadas **declaraciones de marcado**, las cuales pueden ser internas o externas. Normalmente un documento XML se compone de una mezcla de declaraciones de marcado internas y externas. En este último caso debe expresarse en el documento dónde encontrar las declaraciones, así como indicar en la declaración de XML que el documento no es autónomo. Las diferencias entre estos tipos de declaraciones de marcado dan lugar a dos subconjuntos el interno y el externo, conviene saber que primero se procesa el subconjunto interno y después el externo, lo que permite sobrescribir declaraciones externas compartidas entre varios documentos y ajustar el DTD a un documento específico.

- **Subconjunto interno**: Contiene las **declaraciones que pertenecen exclusivamente a un documento** y no es posible compartirlas. Se localizan dentro de unos corchetes que siguen a la declaración de tipo del documento.
- **Subconjunto externo**: Están localizadas en un documento con extensión **dtd** que puede situarse en el mismo directorio que el documento XML. Habitualmente son declaraciones que pueden ser compartidas entre múltiples documentos XML que pertenecen al mismo tipo. En este caso la declaración de documento autónomo ha de ser negativa, ya que es necesario el fichero del subconjunto externo para la correcta interpretación del documento. Con ello el procesado del documento será más lento, ya que antes de procesar el documento el procesador ha de obtener todas las entidades.

    - **<!DOCTYPE nombre_ejemplar SYSTEM "URI"**
      En este caso, se especifica un URI donde pueden localizarse las declaraciones.

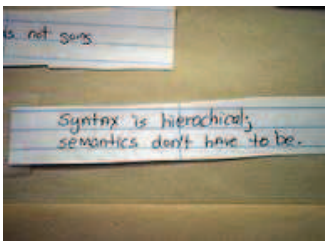
    - **<!DOCTYPE nombre_ejemplar PUBLIC "id_publico" "URI"**
      En este caso también se especifica un identificador, que puede ser utilizado por el procesador XML para intentar generar un URI alternativo, posiblemente basado en alguna tabla. **Como se puede observar también es necesario incluir algún URI.**

  Ahora los corchetes pierden sentido, para localizar las declaraciones del tipo de documento externo mediante una declaración explícita de subconjunto externo se utiliza:

Definición de la sintaxis de documentos XML.

Recordamos que en estos documentos las etiquetas de marcado describen la estructura del documento.

Un elemento es un grupo formado por una etiqueta de apertura, otra de cierre y el contenido que hay entre ambas.



En los documentos de lenguajes de marcas, la distribución de los elementos está jerarquizada según una estructura de árbol, lo que implica que es posible anidarlos pero no entrelazarlos.

Hemos visto que en los elementos el orden es importante, ¿lo es también para los atributos? En este caso el orden no es significativo. Lo que hay que tener presente es que no puede haber dos atributos con el mismo nombre.

Sabemos que los atributos no pueden tener nodos que dependan de ellos, por tanto solo pueden corresponder con hojas de la estructura de árbol que jerarquiza los datos. ¿Significa esto que todas las hojas van a ser atributos? Pues no, es cierto que los atributos son hojas, pero las hojas pueden ser atributos o elementos.

En ese caso, ¿qué criterios podemos utilizar para decidir si un dato del documento que se pretende estructurar ha de representarse mediante un elemento o un atributo? Aunque no siempre se respetan, podemos usar los siguientes criterios:

- **El dato será un elemento si cumple alguna de las siguientes condiciones:**
    - Contiene subestructuras.
    - Es de un tamaño considerable.
    - Su valor cambia frecuentemente.
    - Su valor va a ser mostrado a un usuario o aplicación.
- **Los casos en los que el dato será un atributo son:**
    - El dato es de pequeño tamaño y su valor raramente cambia, aunque hay situaciones en las que este caso puede ser un elemento.
    - El dato solo puede tener unos cuantos valores fijos.
    - El dato guía el procesamiento XML pero no se va a mostrar.

Los espacios de nombres, o namespaces, ¿qué nos permiten?

- Diferenciar entre los elementos y atributos de distintos vocabularios con diferentes significados que comparten nombre.
- Agrupar todos los elementos y atributos relacionados de una aplicación XML para que el software pueda reconocerlos con facilidad.

¿Cómo se declaran?

xmnls:"URI_namespace"

¿Y si se usa un prefijo que nos informe sobre cuál es el vocabulario al que está asociada esa definición?

xmnls:prefijo="URI_namespace"

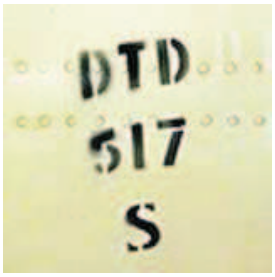
En ambos casos URI_namespace es la localización del conjunto del vocabulario del espacio de nombres al que se hace referencia.

Definiciones de tipo de documento, DTD.

Caso práctico

Según Juan el método más sencillo para intentar normalizar los documentos con los que trabajan, consiste en **definir unos vocabularios** que han de cumplir los documentos que generan, estos se llaman Definición de Tipo de Documento. Además, aunque no es un lenguaje XML, tiene una sintaxis sencilla y fácil para que ella y Félix puedan comprenderla y utilizarla.

Están formadas por una **relación precisa de qué elementos pueden aparecer en un documento y dónde, así como el contenido y los atributos del mismo. Garantizan que los datos del documento XML cumplen las restricciones que se les haya impuesto en el DTD**, ya que estas últimas permiten:



- Especificar la estructura del documento.
- Reflejar una restricción de integridad referencial mínima utilizando (ID e IDREF).
- Utilizar unos pequeños mecanismos de abstracción comparables a las macros, que son las entidades.
- Incluir documentos externos.

¿Cuáles son los inconvenientes de los DTD?
Los principales son:

- Su sintaxis no es XML.
- No soportan espacios de nombres.
- No definen tipos para los datos. Solo hay un tipo de elementos terminales, que son los datos textuales.
- No permite las secuencias no ordenadas.
- No es posible formar claves a partir de varios atributos o elementos.
- Una vez que se define un DTD no es posible añadir nuevos vocabularios.

Cuando están definidas dentro del documento XML se ubican entre corchetes después del nombre del ejemplar en el elemento <!DOCTYPE> pero, cuando está definido en un fichero externo ¿a qué tipo de fichero corresponde?
Definimos el DTD externo en un fichero de texto plano con extensión **dtd**.

Declaraciones de tipos de elementos terminales.

Los **tipos terminales** son aquellos elementos que se corresponden con hojas de la estructura de árbol formada por los datos del documento XML asociado al DTD. La declaración de tipos de elementos está formada por la cadena **"<!ELEMENT"** separada por, al menos un espacio del nombre del elemento XML que se declara, y seguido de la declaración del contenido que puede tener dicho elemento.

En el caso de elementos terminales, es decir, aquellos que no contienen más elementos, esta declaración de contenido es dada por uno de los siguientes valores:

- **EMPTY:** Indica que el elemento no es contenedor. Por ejemplo, la siguiente definición muestra un elemento A que no contiene nada:
  <!ELEMENT A EMPTY>
- **ANY:** Permite que el contenido del elemento sea cualquier cosa. Un ejemplo de definición de un elemento de este tipo es:
  <!ELEMENT A ANY>
- **(#PCDATA):** Indica que los datos son analizados en busca de etiquetas, resultando que el elemento no puede contener elementos, es decir solo puede contener datos de tipo carácter exceptuando los siguientes: <, &,]], >. Si es de este tipo, el elemento A tendrá una definición como:
  <!ELEMENT A (#PCDATA)>

Declaraciones de tipos de elementos no terminales.

Una vez que sabemos el modo de definir las hojas de un árbol de datos **veamos cómo definir sus ramas**, es decir los elementos que están formados por otros elementos.
Para definirlos utilizamos referencias a los grupos que los componen tal y como muestra el ejemplo:

<!ELEMENT A (B, C)>

En este caso se ha definido un elemento A que está formado por un elemento B seguido de un elemento C.

¿Y qué sucede cuando un elemento puede aparecer en el documento varias veces, hay que indicarlo de algún modo? Pues sí, también hay que indicar cuando un elemento puede no aparecer. Para ello usamos los siguientes operadores, que nos permiten definir la cardinalidad de un elemento:

- **Operador opción, ?.** Indica que el elemento no es obligatorio. En el siguiente ejemplo el subelemento trabajo es opcional.
  <!ELEMENT telefono (trabajo?, casa)

- **Operador uno-o-más, +.** Define un componente presente al menos una vez. En el ejemplo definimos un elemento formado por el nombre de una provincia y otro grupo, que puede aparecer una o varias veces.
  <!ELEMENT provincia (nombre, (cp, ciudad)+)

- **Operador cero-o-mas, *.** Define un componente presente cero, una o varias veces. En el ejemplo el grupo (cp, ciudad) puede no aparecer o hacerlo varias veces.
  <!ELEMENT provincia (nombre, (cp, ciudad)*)

- **Operador de elección, |.** Cuando se utiliza sustituyendo las comas en la declaración de grupos indica que para formar el documento XML hay que elegir entre los elementos separados por este operador. En el ejemplo siguiente, el documento XML tendrá elementos provincia que estarán formados por el elemento nombre y el cp (código postal), o por el elemento nombre y la ciudad.
  <!ELEMENT provincia (nombre, (cp | ciudad))

Ejercicio resuelto

Creación de un DTD correspondiente a la siguiente estructura de datos de un documento XML:

```
<alumno>
   <nombre>Olga</nombre>
   <dirección>El Percebe 13</dirección>
</alumno>
```

Declaraciones de listas de atributos para los tipos de elementos.

Ya sabemos cómo declarar elementos, ahora veamos el modo de **declarar los atributos asociados a un elemento.** Para ello utilizamos la cadena <!ATTLIST seguida del nombre del elemento asociado al atributo que se declara, luego el nombre de éste último seguido del tipo de atributo y del modificador. Este elemento puede usarse para declarar una lista de atributos asociada a un elemento, o repetirse el número de veces necesario para asociar a dicho elemento esa lista de atributos, pero individualmente.

Al igual que los elementos no todos los atributos son del mismo tipo, los más destacados son:

- Enumeración, es decir, el atributo solo puede tomar uno de los valores determinados dentro de un paréntesis y separados por el operador |.
  <!ATTLIST fecha dia_semana (lunes|martes|miércoles|jueves|viernes|sábado|domingo) #REQUIRED>

- **CDATA**, se utiliza cuando el atributo es una cadena de texto.
- **ID**, permite declarar un atributo identificador en un elemento. Hay que recordar que este valor ha de ser único en el documento. Además hay que tener en cuenta que los números no son nombres válidos en XML, por tanto no son un identificador legal de XML. Para resolverlo suele incluirse un prefijo en los valores y separarlo con un guión o una letra.
- **IDREF**, permite hacer referencias a identificadores. En este caso el valor del atributo ha de corresponder con el de un identificador de un elemento existente en el documento.
- **NMTOKEN**, permite determinar que el valor de un atributo ha de ser una sola palabra compuesta por los caracteres permitidos por XML.

¿También hemos de declarar si el valor de un atributo es obligatorio o no? Si, para ello se usan los siguientes modificadores:

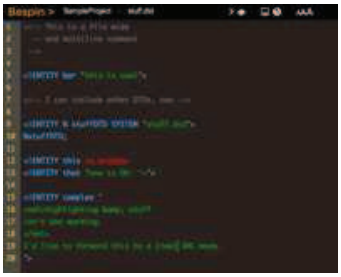
- **#IMPLIED**, determina que el atributo sobre el que se aplica es opcional.
- **#REQUIRED**, determina que el atributo tiene carácter obligatorio.
- **#FIXED**, permite definir un valor fijo para un atributo independientemente de que ese atributo se defina explícitamente en una instancia del elemento en el documento XML.
- Literal, asigna a un atributo el valor dado por una cadena entre comillas.

Ejercicio resuelto

Creación de un DTD correspondiente a la siguiente estructura de datos de un documento XML:

```
<alumno edad=15>
   <nombre>Olga</nombre>
   <apellidos>Velarde Cobo</apellidos>
   <dirección>El Percebe 13</dirección>
</alumno>
```

Declaraciones de entidades.



¿Qué sucede si queremos **declarar valores constantes dentro de los documentos**? ¿podemos?
Las entidades nos permiten definir constantes en un documento XML. Cuando se usan dentro del documento XML se limitan por "&" y ";", por ejemplo &entidad;
¿Cómo trabaja el intérprete con ellos? Al procesar el documento XML, el intérprete sustituye la entidad por el valor que se le ha asociado en el DTD.
No admiten recursividad, es decir, una entidad no puede hacer referencia a ella misma.
Para definir una entidad en un DTD se usa el elemento <!ENTITY>

Las entidades pueden ser de tres tipos:

- **Internas:** Existen cinco entidades predefinidas en el lenguaje, son:
  - **<:** Se corresponde con el signo menor que, <.
  - **>:** Hace referencia al signo mayor que, >.
  - **":** Son las comillas rectas dobles, ".
  - **':** Es el apóstrofe o comilla simple, '.
  - **&:** Es el et o ampersand, &.

¿Se puede definir una entidad diferente? ¿Cómo?
Utilizando la siguiente sintaxis:

<!ENTITY nombre_entidad "valor de la entidad">

Por ejemplo, <!ENTITY dtd "Definiciones de Tipo de Documento">

- **Externas:** Permiten establecer una relación entre el documento XML y otro documento a través de la URL de éste último. Un ejemplo de declaración de una entidad externa es:
<!ENTITY nombre_entidad SYSTEM "http://localhost/docsxml/fichero_entidad.xml">

En este caso el contenido de los ficheros es analizado, por lo que deben seguir la sintaxis XML.

Cuando es necesario incluir ficheros con formatos binarios, es decir ficheros que no se analicen, se utiliza la palabra reservada NDATA en la definición de la entidad y habrá que asociar a dicha entidad una declaración de notación, tal y como muestra el ejemplo del apartado siguiente.

- **De parámetro:** Permite dar nombres a partes de un DTD y hacer referencia a ellas a lo largo del mismo. Son especialmente útiles cuando varios elementos del DTD comparten listas de atributos o especificaciones de contenidos. Se denotan por %entidad;
<!ENTITY %direccion "calle, numero?, ciudad, cp">
<!ENTITY alumno (dni, %direccion;)>
<!ENTITY ies (nombre, %direccion;)>

- **De parámetro externas:** Permite incluir en un DTD elementos externos, lo que se aplica en dividir la definición DTD en varios documentos.
<!ENTITY persona SYSTEM "persona.dtd">

Declaraciones de notación.

Cuando se incluyen ficheros binarios en un fichero, ¿cómo le decimos qué aplicación ha de hacerse cargo de ellos? La respuesta es utilizando notaciones. La sintaxis para declarar **notaciones** es:

<!NOTATION nombre SYSTEM aplicacion>

Por ejemplo, una notación llamada **gif** donde se indica que se hace referencia a un editor de formatos gif para visualizar imágenes será:

<!NOTATION gif SYSTEM "gifEditor.exe">

Para asociar una entidad externa no analizada, a esta notación basta declarar dicha entidad del siguiente modo:

<!ENTITY dibujo SYSTEM "imagen.gif" NDATA gif>

Secciones condicionales.

Permiten incluir o ignorar partes de la declaración de un DTD. Para ello se usan dos tokens:

- **INCLUDE**, permite que se vea esa parte de la declaración del DTD. Su sintaxis es:
<![INCLUDE [Declaraciones visibles]] >

  Por ejemplo:

  <![INCLUDE [<!ELEMENT nombre (#PCDATA)>]] >

- **IGNORE**, permite ocultar esa sección de declaraciones dentro del DTD. La forma de uso es:
<![IGNORE [Declaraciones ocultas]] >

  Por ejemplo: <![IGNORE [<!ELEMENT clave (#PCDATA)>]] >

XML Schema.

Caso práctico

Félix, quien considera que la normalización de los documentos XML que manejan en la empresa va a ser un duro trabajo para María, él y otros trabajadores inexpertos, plantea la posibilidad de que se encargue de ello algún trabajador de la **consultoría informática que dirige Juan.**
Al final se va a encargar de ello **Marina.** Les explica que, en lugar de trabajar con DTD's le parece mejor hacerlo con un lenguaje XML llamado **XML Schema**, el cual tiene, entre otras, la ventaja de permitir definir el tipo de datos de cada uno de los componentes de cada documento.

Los **DTD** permiten diseñar un **vocabulario para ficheros XML**, pero, ¿qué sucede cuando los valores de los elementos y atributos de esos ficheros han de corresponder a datos de un tipo determinado, o cumplir determinadas restricciones que no pueden reflejarse en los DTD? Para ello se definen **XML Schemas.**

¿También **se definen en ficheros planos**? Si, ya que son documentos XML, pero en este caso la extensión de los archivos es xsd, motivo por el cual también se les denomina documentos XSD.

Los elementos XML que se utilizan para generar un esquema han de pertenecer al espacio de nombre XML Schema, que es: http://www.w3.org/2001/XMLSchema.

El ejemplar de estos ficheros es **<xs:schema>**, contiene declaraciones para todos los elementos y atributos que puedan aparecer en un documento XML asociado válido. Los elementos hijos inmediatos de este ejemplar son **<xs:element>** que nos permiten crear globalmente un elemento. Esto significa que el elemento creado puede ser el ejemplar del documento XML asociado.

Debes conocer

En este primer enlace encontrarás los fundamentos del estándar XML Schema.

XML Schema Fundamentos

XML Schema Tipos de datos

Tipos de datos.

Son los distintos valores que puede tomar el **atributo type** cuando se declara un elemento o un atributo y representan el tipo de dato que tendrá el elemento o atributo asociado a ese **type** en el documento XML.

Algunos de estos valores predefinidos son:

- **string**, se corresponde con una cadena de caracteres UNICODE.
- **boolean**, representa valores lógicos, es decir que solo pueden tomar dos valores, true o false.
- **integer**, número entero positivo o negativo.
- **positiveInteger**, número entero positivo.
- **negativeInteger**, número entero negativo.
- **decimal**, número decimal, por ejemplo, 8,97.
- **dateTime**, representa una fecha y hora absolutas.
- **duration**, representa una duración de tiempo expresado en años, meses, días, horas, minutos segundos. El formato utilizado es: PnYnMnDTnHnMnS. Por ejemplo para representar una duración de 2 años, 4 meses, 3 días, 5 horas, 6 minutos y 10 segundos habría que poner: P2Y4M3DT5H6M7S. Se pueden omitir los valores nulos, luego una duración de 2 años será P2Y. Para indicar una duración negativa se pone un signo – precediendo a la P.
- **time**, hora en el formato hh:mm:ss.
- **date**, fecha en formato CCYY-MM-DD.
- **gYearMonth**, representa un mes de un año determinado mediante el formato CCYY-MM.
- **gYear**, indica un año gregoriano, el formato usado es CCYY.
- **gMothDay**, representa un día de un mes mediante el formato –MM-DD.
- **gDay**, indica el ordinal del día del mes mediante el formato –DD, es decir el 4º día del mes será –04.
- **gMonth**, representa el mes mediante el formato –MM. Por ejemplo, febrero es –02.
- **anyURI**, representa una URI.
- **language**, representa los identificadores de lenguaje, sus valores están definidos en RFC 1766.
- **ID, IDREF, ENTITY, NOTATION, MTOKEN.** Representan lo mismo que en los DTD's (ver apartado 2.3).

Facetas de los tipos de datos.

¿Cuáles son las restricciones que podemos aplicar sobre los valores de los datos de un elemento o atributo? Están definidos por las **facetas**, que solo pueden aplicarse sobre tipos simples utilizando el elemento **xs:restriction**. Se expresan como un elemento dentro de una restricción y se pueden combinar para lograr restringir más el valor del elemento. Son, entre otros:

- **length, minlength, maxlentgh:** Longitud del tipo de datos.
- **enumeration:** Restringe a un determinado conjunto de valores.
- **whitespace:** Define el tratamiento de espacios (preserve/replace, collapse).

- **(max/min)(In/Ex)clusive:** Límites superiores/inferiores del tipo de datos. Cuando son Inclusive el valor que se determine es parte del conjunto de valores válidos para el dato, mientras que cuando se utiliza Exclusive, el valor dado no pertenece al conjunto de valores válidos.
- **totalDigits, fractionDigits:** número de dígitos totales y decimales de un número decimal.
- **pattern:** Permite construir máscaras que han de cumplir los datos de un elemento. La siguiente tabla muestra algunos de los caracteres que tienen un significado especial para la generación de las máscaras.

Elementos para hacer patrones.

Patron	Significado
[A-Z a-z]	Letra.
[A-Z]	Letra mayúscula.
[a-z]	Letra minúscula.
[0-9]	Dígitos decimales.
\D	Cualquier carácter excepto un dígito decimal.
(A)	Cadena que coincide con A.
A | B	Cadena que es igual a la cadena A o a la B.

Elementos para hacer patrones.

Patron	Significado
AB	Cadena que es la concatenación de las cadenas A y B.
A?	Cero o una vez la cadena A.
A+	Una o más veces la cadena A.
A*	Cero o más veces la cadena A.
[abcd]	Alguno de los caracteres que están entre corchetes.
[^abcd]	Cualquier carácter que no esté entre corchetes.
\t	Tabulación.

Elementos del lenguaje.

Algunos de los más usados son:

- Esquema, **xs:schema**, contiene la definición del esquema.
- Tipos complejos, **xs:complexType**, define tipos complejos.
- Tipos simples, **xs:simpleType**, permite definir un tipo simple restringiendo sus valores.
- Restricciones, **xs:restriction**, permite establecer una restricción sobre un elemento de tipo base.
- Agrupaciones, **xs:group**, permite nombrar agrupaciones de elementos y de atributos para hacer referencia a ellas.
- Secuencias, **xs:secuence**, permite construir elementos complejos mediante la enumeración de los que les forman.
- Alternativa, **xs:choice**, representa alternativas, hay que tener en cuenta que es una o-exclusiva.
- Contenido mixto, definido dando valor true al atributo mixed del elemento **xs:complexType**, permite mezclar texto con elementos.
- Secuencias no ordenadas, **xs:all**, representa a todos los elementos en cualquier orden.

Debes conocer

Este enlace te permitirá consultar las estructuras del estándar XML Schema.

XML Schema Estructuras

Definición de tipos de datos XML Schema.

En los DTD se diferencia entre los elementos terminales y los no terminales ¿en este caso también? Si, este lenguaje **permite trabajar tanto con datos simples como con estructuras de datos complejos**, es decir, compuestos por el anidamiento de otros datos simples o compuestos.

- **Tipos de datos simples.**
  Estos datos se suelen definir para hacer una restricción sobre un tipo de datos XDS ya definido y establece el rango de valores que puede tomar.

También se pueden crear tipos de datos simples basados en listas de valores utilizando el atributo derivedBy de simpleType.

- **Tipos de datos compuestos.**
  El elemento xsd:complexType permite definir estructuras complejas de datos. Su contenido son las declaraciones de elementos y atributos, o referencias a elementos y atributos declarados de forma global. Para determinar el orden en que estos elementos aparecen en el documento XML se utiliza el elemento .

Asociación con documentos XML.

Una vez que tenemos creado el **fichero XSD** ¿cómo lo **asociamos a un fichero XML**?

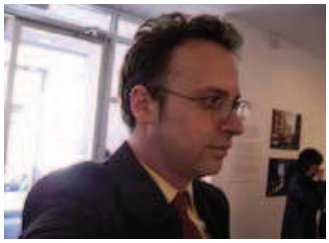
El modo de asociar un esquema a un documento XML es un espacio de nombres al ejemplar del documento, donde se indica la ruta de localización de los ficheros esquema mediante su URI, precedida del prefijo " **xsi:**".

Herramientas de creación y validación.

Caso práctico

Antes de comenzar a trabajar con la **normalización de los documentos** que utiliza **la empresa de María y Félix. Marina** le presenta a **Juan** un informe sobre las diferentes herramientas que pueden facilitarles el trabajo de edición y validación de los documentos **XSD** y **XML**.

Juan hará un estudio de costes y escogerá alguna de ellas para realizar el trabajo.



Igual que hasta ahora, para crear y validar los documentos XML y los esquemas basta con un editor de texto plano y un navegador. ¿Pero no hay ninguna herramienta que nos facilite el trabajo? Pues sí, existen aplicaciones que permiten al usuario visualizar, validar y editar documentos en el lenguaje XML. Algunos de estos productos son:

- Editix XML Editor (Gratuito).
- Microsoft Core XML Services (MSXML) (Gratuito).
- XMLFox Advance.
- Altova XML Spy Edición Estándar.
- Editor XML xmlBlueprint.
- Editor Gráfico XSD y XML (XML Studio) (Gratuito).
- Estudio XML Líquido (Gratuito).
- Stylus Studio 2001 (Gratuito).
- Oxygen XML Editor.
- Exchanger XML Editor.