El Metalenguaje XML y el Esquema de Tipo de Elemento

JUAN VOUTSSÁS MÁRQUES
Centro Universitario de Investigaciones Bibliotecológicas
de la UNAM, 04510, México D.F., Tel: 56-23-03-29
E-Mail: voutssás@servidor.unam.mx

Artículo

RESUMEN

En las últimas décadas se ha ido desarrollando una herramienta para el procesamiento de información electrónica conocida como los metalenguajes de marcado. Desde el formato "MARC" de los sesentas hasta el "XML" establecido en nuestros días ha habido sin duda una evolución en este sentido bastante notable. El presente documento trata de establecer una evolución histórica y funcional sobre los lenguajes de marcado, sus características, ventajas, limitaciones y sus expectativas.

Dentro de las especificaciones "XML" se ha desarrollado recientemente una técnica particular de declaración y manejo de la información documental conocida como "esquema de tipo de elemento" o simplemente esquemas, la cual va más allá de la simple y ya conocida definición del documento a través de un descriptor de tipo de documento (DTD). Los esquemas nos ofrecen una posibilidad mucho más amplia para definir clases de documentos. Este trabajo intenta establecer esas posibilidades para poder entender la potencialidad de los esquemas, su estado del arte, sus tendencias y finalmente la importancia que el desarrollo de esquemas especiales para ciertas clases de documentos pueden tener en nuestro medio bibliotecario mexicano para un mejor establecimiento y explotación de las colecciones digitales.

Palabras Clave: Metalenguajes, Lenguajes de marcado documental, XML, HTML, SGML, MARC, DTD, Esquema de tipo de elemento, Esquemas, Bibliotecas digitales, Desarrollo de colecciones digitales, Tecnologías de información

METALANGUAGE XML AND THE TYPE OF ELEMENT SCHEME

JUAN VOUTSSÁS-MÁRQUES

ABSTRACT

During the last decades a tool for electronic processing known as mark-up languages has been and is still being developed. From the MARC Format of the 60's to the XML established in our days, a very clear evolution has, no doubt, taken place. This paper intends to settle the issue of the historical and functional evolution of the mark-up languages and of their characteristics, advantages, limitations and expectations.

Within the specifications of XML a particular technique for the declaration and handling of the documentary information has been developed recently. The technique is known as Type of Element Scheme, or simply Schemes, but it goes well beyond the simple and very well known definition of document through a Descriptor of Type of Document (DTD). The schemes offer a much wider possibility for defining classes of documents. This paper attempts to establish these possibilities in order to understand the potentiality of Schemes, its state of art, its tendencies and also the importance that the development of special schemes may have for certain classes of documents in our Mexican library environment for a better implementation and exploitation of digital collections.

Key Words: Metalanguages; Document Markup Languages; XML; HTML; SGML; MARC; DTD; Esquemas; Digital Library; Digital Collection Development; Information Technology

ANTECEDENTES

Desde la década de los sesentas, el personal técnico relacionado con el intercambio de documentos por medio de una computadora comenzó a preocuparse por estructurar esa información en una forma normalizada que facilitara el intercambio y la manipulación de esos materiales. En el ámbito de las bibliotecas surgió el proyecto del formato "MARC", *(Machine Readable Cataloguing)*, pionero en esta temática. Durante 1965 la Biblioteca del Congreso de los Estados unidos (LC) estableció un proyecto piloto denominado "MARC I" cuyo objetivo era definir una metodología para crear registros catalográficos en un formato legible **por una computadora**. Un proyecto similar apareció en el Reino Unido en esa misma época ordenado por el Consejo de la Biblioteca Británica (British Library) con el fin de establecer un formato para registrar y explotar los registros que serían usados en la Bibliografía Británica. A este proyecto se le llamó "BNB MARC" *(British National Bibliography with MAchine Readable Cataloguing)*. Ambos proyectos derivaron en poco tiempo en una cooperación en el ambiente de las bibliotecas anglo-sajonas y dieron origen, en 1968, al proyecto "MARC II". El resultado fue el formato ampliamente conocido para intercambiar registros catalográficos de monografías vía electrónica, el cual fue complementado poco después con los consecuentes formatos para publicaciones periódicas, mapas, discos, etcétera, hasta completar prácticamente todo los tipos de documentos que existen en las bibliotecas.

Con el tiempo MARC fue tan exitoso y tan ampliamente utilizado en todo el mundo que empezaron a usarse múltiples variantes del mismo, hasta llegar a unas veinte que fueron adaptadas a las nuevas necesidades de otras comunidades en el planeta. Para 1977 se definió el formato universal "UNIMARC" como una especie de "esperanto" para MARC con objeto de que los distintos "MARCs" pudieran intercambiar registros entre sí usando "UNIMARC" como intermediario. Además el formato también

empezó a ser definido para otras entidades de datos más allá de la simple producción de catálogos, como los registros de autoridad, pues su diseño permitía que fuera usado en prácticamente cualquier plataforma de cómputo del mundo. Debido a su amplia aceptación MARC se convirtió en norma ANSI para la unión americana en 1971 (Z39.2 - 1971) y en norma ISO internacional en 1973 (ISO 2709: 1973 E). Está de más ahondar sobre lo que MARC representó para el registro electrónico de información. [*Library of Congress*, 2003]

En esencia, MARC no ha sido nunca un catálogo ni un método para catalogar, como muchos han pensado a lo largo del tiempo. La **gran aportación de MARC al manejo documental**, y que ha sido seguida por muchos formatos posteriormente hasta nuestros días, fue la de asociar una "etiqueta" a cada uno de los elementos que conforman una ficha catalográfica, con objeto de que las computadoras puedan identificar y catalogar sus partes y posteriormente efectuar múltiples acciones con ellas en beneficio de las personas. Este principio tan simple y tan obvio en nuestra época marcó un hito en la historia del manejo documental por medio de computadoras. Hoy se ha sofisticado y diversificado enormemente, pero el principio se mantiene hasta nuestros días. [A general introduction to marc] [Iflanet. *What is marc*]

De hecho, este concepto de asociar cada elemento de una ficha, y posteriormente de cualquier otro documento, con una "etiqueta" (llamada en inglés *label, tag* o *token*) fue conociéndose como "marcado" del documento, y ha sido de tal trascendencia que todos los formatos que han aparecido posteriormente para el registro de documentos se basan en el concepto del marcado, y así lo ostentan en su nombre *(markup o mark up.)* Abundaré más adelante en este concepto. Por lo anterior, MARC puede ser considerado el "abuelo" de todos los lenguajes de marcado documental para computadora que existen hoy.

Fuera de las bibliotecas, a nivel general y por esa misma época de fines de los sesentas, la empresa IBM creó "GML" *(Generalized Markup Language)* o "Lenguaje de marcado generalizado" para contender con las necesidades de sus propios sistemas internos de publicación. Ellos usaron GML para producir de manera normalizada, libros, reportes, manuales y otros tipos de documentos a partir de un único conjunto de archivos originales en una computadora. Muchas otras soluciones fueron introducidas por otras organizaciones para estructurar información documental, pero nada realmente que abarcara este propósito a gran escala.

La primera tecnología de estructuración de información documental normalizada de cierta significancia fue el "SGML" *(Standard Generalized Markup Language)* o "Lenguaje de marcado generalizado estándar", en lo sucesivo "SGML", el cual también provino de la empresa IBM a principios de los ochentas. Este lenguaje fue creado con el fin de formatear y organizar la documentación legal dentro de esa empresa, pero posteriormente fue expandido y adaptado para ser usado en una amplia variedad de empresas como un estándar para manejar todo tipo de información y para 1986 se convirtió en una norma ISO [ISO 8879, 1986].

Si bien SGML es extremadamente poderoso, es igualmente complejo y requiere de una considerable cantidad de programación adicional para procesarlo. Debido a su complejidad y los recursos extra que demanda, SGML no era una opción viable para representar hipertextos en las primeras épocas de la Internet, dadas las dimensiones y limitaciones de los equipos de esa época.

En 1989, Tim Berners-Lee y Anders Berglund, dos investigadores del Laboratorio Europeo de Partículas Físicas (CERN), crearon un lenguaje basado en etiquetas para marcar documentos técnicos y serés tos compartidos en Internet. Este lenguaje fue expandido en 1990 a una versión simplificada del SGML llamada HTML *(Hyper-Text Markup Language)* "Lenguaje de marcado de hipertextos", y ha llegado a ser el formato estándar para el manejo de información en la "*Web*".

HTML podía representar perfectamente información *estática* en una página Web; es decir, textos previamente establecidos, imágenes, botones, ligas, etcétera, pero pronto hubo necesidad de ligar las páginas a bases de datos con el fin de traer a las pantallas catálogos, estados de cuenta, textos completos, etcétera; es decir, información *dinámica* o cambiante, para lo cual HTML no fue diseñado. Algunos añadidos han surgido para HTML en los últimos años, para resolver esta problemática, algunos de ellos muy exitosos, como los derivados del lenguaje "Java", y otros han caído ya en el olvido. Al final, HTML comienza a estar demasiado "remendado" y resulta ya complejo para poder satisfacer los requerimientos del manejo de información de la actualidad. [Teasdale, 1995]

Como resultado de esto muchos técnicos han empezado a volver sus ojos nuevamente hacia SGML, ya que su complejidad era de origen y no resultado de adiciones, y siempre ha tenido capacidad para manejar documentos complejos de variados tipos. Otros grupos de personas empezaron a reescribir una versión simplificada de SGML pero capaz de contender con las carencias de HTML; de ahí surge un nuevo lenguaje llamado XML: el (*eXtensible Markup Language)* o "Lenguaje de marcado extensible", en lo sucesivo "XML". En 1996, el "Consorcio para el desarrollo de la World Wide Web", conocido como el W3C, sentó las bases para estos desarrollos. Se establecieron las ventajas propias del SGML: estructura, extensibilidad y validación, y se creó un grupo de trabajo que estableciera las bases para un nuevo lenguaje de marcado que conservara las ventajas centrales del SGML y que tuviera la simplicidad del HTML; esto es, un SGML "aerodinámico" para la Web. El resultado fue que en 1998 se publicaron las especificaciones de la primera versión del XML [Extensible Markup Language 1.0, 2000]. En cuanto a dimensiones, la especificación XML resultó ser de menos de una décima parte de la de SGML, con lo cual puede estimarse el grado de compactación. [MSDN on Line, 2000]. Desde ese año a la fecha, el consorcio W3C se ha constituido como la instancia oficial a nivel mundial para definiciones del estándar XML.

CONCEPTOSBÁSICOS

Metalenguaje

Si se observa la literatura al respecto, puede notarse que HTML, SGML, XML y otros entes del manejo de información documental son denominados como "lenguajes", pero más propiamente dicho, se trata de "metalenguajes". ¿Qué significa realmente este término? es importante aclararlo para poder comprender el contexto de esas herramientas.

El concepto de "metalenguaje" no es simple y pretende establecer las reglas por las cuales un lenguaje dado puede ser definido o examinado **formalmente**, pero existen muchas clases de lenguajes: los lenguajes que los humanos usamos para comunicarnos cotidianamente, español, inglés, francés, etcétera; lenguajes para programar una computadora, Fortran, Algol, "C", Pascal, Cobol, Java, etcétera; el álgebra, etcétera.

Así pues existen muchos metalenguajes; más de los lenguajes que pueden ser creados. Incluso un lenguaje cualquiera que hablamos las personas incluye varios "sublenguajes" o "argots" propios de una profesión o clase social. Distinguimos por ejemplo el lenguaje propio de los médicos dentro de un lenguaje dado, o el del personal técnico de cómputo, etcétera. Los enfoques pueden ser entonces filológicos, matemáticos, computacionales, etcétera. Para fines de este trabajo nos centraremos en los lenguajes "documentales", que son los de nuestro interés.

De esta forma podemos establecer un metalenguaje para definir un lenguaje dado y por tanto, **definiremos entonces de una forma simple "metalenguaje" como un medio para definir formalmente un lenguaje dado**. Si bien entran también aquí consideraciones de sintaxis, contexto, objetos, etcétera, concentrándonos en el tema que nos ocupa; es decir los documentos, resulta que los metalenguajes (MARC, SGML, HTML, XML) son un medio para describir un lenguaje de marcado; dicho de otra forma, son metalenguajes para definir formalmente un lenguaje de marcado de documentos.

Las reglas para definir el marcado de documentos, su sintaxis, sus limitaciones y características están dentro de cada uno de esos entes (MARC, SGML, HTML, XML); es decir, ellos internamente van conformando todo un lenguaje para marcar o codificar documentos. El lenguaje está por lo tanto intrínsecamente contenido en cada uno de ellos, pero la manera de presentar el conjunto de las reglas y definiciones de ese lenguaje es un metalenguaje. Por eso decimos que estas herramientas son metalenguajes más propiamente dicho que lenguajes.

Históricamente, la palabra marcado (*markup*) viene de las instrucciones que se le daban a un impresor o tipógrafo sobre cómo debería imprimirse un cierto pasaje de un texto: las negritas, los subrayados, la tipografía, el tipo y el tamaño de letra, los símbolos especiales, etcétera, se señalaban con anotaciones o "marcas" al margen del texto; de ahí el nombre. Con forme los textos se fueron automatizando, el término se fue extendiendo hasta cubrir toda clase de códigos de marcado insertados en esos

textos electrónicos con objeto de incrustar una serie de características propias de cada sección o parte del texto.

De este concepto se desprende que el marcado es una codificación del texto, gracias a la cual cada porción del mismo se va haciendo explícita y toma forma en un contexto. De hecho, y hablando formalmente, todos los textos han tenido desde hace largo tiempo una manera de hacerse explícitos y entendibles al lector. Los signos de puntuación, los acentos, los espacios entre palabras, el uso de mayúsculas, los párrafos, los capítulos, los incisos, etcétera, se han agregado a los textos desde hace mucho tiempo para que sus lectores puedan leer explícitamente un conjunto de letras y darles un sentido. Imagínese el lector un texto en el cual todas las letras están contiguas, sin espacios ni puntuaciones ni nada adicional, y podrá darse cuenta de la enorme ayuda que estos elementos proporcionan para hacer explícitas estas palabras. Si el lector no se lo imagina analice entonces este texto:

adanyrazaazarynada

Como puede observarse, el texto anterior difícilmente puede ser interpretado al primer golpe de vista; es necesario leerlo y releerlo varias veces para que el cerebro humano, tan avezado en este tipo de actividades, pueda establecer patrones, cortes, analogías, y descifrar su significado y su contexto. Puesto de esta forma:

palíndromo:

Adán y raza, azar y nada

es mucho más fácil de interpretar. Ciertas convenciones de espacios y puntuación lo hacen explícito. Incluso, gracias al encabezado los lectores se cercioraron de que el texto es un palíndromo, es decir, algo que puede ser leído igualmente de izquierda a derecha que de derecha a izquierda. Algunos no se habrán percatado de ello hasta que el encabezado, es decir, cierto *marcado,* lo hizo explícito.

Del mismo modo, y en ello radica su importancia, un lenguaje de marcado debe permitir hacer explícito un texto **para los sistemas y las máquinas** que los operan electrónicamente: además de dónde empieza y termina cada palabra, lo cual puede lograrse con espacios, nos indica dónde empieza una parte de un texto en un documento y dónde acaba, qué partes tiene ese documento y cómo se llaman, etcétera. El lenguaje de marcado es entonces un conjunto de convenciones preestablecidas que se usan de manera conjunta y ordenada para marcar o codificar un texto de modo que éste pueda ser entendido explícitamente por máquinas para que éstas, a su vez, se lo puedan hacer explícito a las personas.

Un lenguaje de marcado debe especificar entonces cuáles marcas son permitidas, cuáles son obligatorias u opcionales, cómo deben diferenciarse esas marcas del texto propiamente dicho y, por supuesto, qué significa cada marca.

Trataré de ilustrar con un ejemplo los temas anteriormente expuestos.

Usted quiere describir una parte de un documento que es la dirección postal de una persona dada, la cual, como sabemos, está formada a su vez por varios elementos. ¿Cómo describiríamos esta entidad en lenguaje llano pero a la vez **formalmente**?

Tal vez sería algo como esto:

> Una dirección postal consiste de: una parte-que-tiene-el-nombre, seguida de una parte-que-tiene-la-dirección, seguida de una parte-que-tiene-el-código-postal.
> La parte-que-tiene-el-nombre consiste de: un nombre-de-pila, seguido de un apellido-paterno, seguido de un apellido-materno, si se tiene.
> La parte-que-tiene-la-dirección consiste de: un nombre-de-calle, seguido de un número-exterior, seguido (si lo hubiese) de un número-interior, seguido de un nombre-de-colonia.
> La parte-que-tiene-el-código-postal consiste de: un nombre-de-ciudad, seguido de un nombre-de-entidad-federativa, seguido de un código-postal
> Un código-postal consiste de un número de cinco dígitos.

Usemos un metalenguaje para describir esto. Por simplicidad, usaremos la "forma de Backus-Naur" (Backus-Naur Form o BNF), un metalenguaje muy utilizado para definir lenguajes de programación desde 1959 y que toma el nombre de sus creadores John Backus y Peter Naur. [*FreeDictionary*....2000].

Las bases de construcción del metalenguaje son las siguientes; primero los símbolos:

- **::=**: significa "se define como"
- **|**: significa "o"
- **<>**: corchetes angulares usados para encerrar nombres de categorías o "entes".
- **{}**: corchetes de llave usados para encerrar nombres de categorías opcionales.
- **" "**: Texto entre comillas que se integra literalmente tal como está escrito.
- **<EOL>**: fin de la línea de definición

La manera de usar la forma de Backus-Naur es la siguiente: se pone del lado izquierdo la entidad a definir y del lado derecho la definición, separadas por el signo de "se define como". Se trata de pasar de lo general a lo particular y se evitará a toda costa la tautología; es decir que la parte a definir sea igual que la definida. No obstante, **una parte** de lo que está a la derecha como definida puede ser parte de lo que se define. Por ejemplo, en el álgebra, la expresión $n = n+1$ no puede existir, ya que de acuerdo a sus reglas, no hay ningún número que sea igual a él mismo más uno. En notación *BNF* $n ::= n+1$ sí puede existir y significa que n se define como él mismo incrementado en una unidad. Perfecta manera de establecer un contador. Si se observa el ejemplo anterior, se ve que *n* forma parte de lo que se define y de lo definido, pero es correcto bajo estas reglas y no es una tautología.

Si en algún momento una definición puede ser definida a su vez en componentes más simples, esa nueva definición se escribe a continuación.

De acuerdo con lo anterior, podemos definir más formalmente una dirección postal según la notación de Backus-Naur como:

```
<dirección-postal> ::= <parte-que-tiene-el-nombre> <parte-que-tiene-la-direc-
ción> <parte-que-tiene-el-código-postal>
<parte-que-tiene-el-nombre> ::= <nombre-de-pila> <apellido-paterno> {apellido-
materno}
<nombre-de-pila> ::= <nombre> | <inicial> "." <EOL>
| <nombre> <nombre-de-pila>
<parte-que-tiene-la-dirección>::= <nombre-de-calle> <número-exterior> {núme-
ro-interior}<nombre-de-colonia>
<parte-que-tiene-el-código-postal> ::= <nombre-de-ciudad><nombre-de-entidad-
federativa> <código-postal>
<código-postal> ::= <dígito> <dígito><dígito><dígito><dígito>
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

De acuerdo con este ejemplo, y utilizando la notación BNF podemos observar lo siguiente:

- Hemos definido dirección-postal como una categoría formada por tres partes (nombre, dirección y código) y éstas deben existir siempre y en ese orden.
- Hemos definido a su vez al nombre como una categoría formada por el nombre-de-pila, seguido por el apellido paterno y opcionalmente, el apellido materno, y en ese orden.
- Hemos definido también al nombre de pila como una categoría que puede estar formada por dos posibilidades: una de ellas es un nombre o una inicial más un punto obligatorio; la otra posibilidad es compuesta, y puede estar formada por un nombre seguido de lo que habíamos ya utilizado como nombre de pila; es decir, el concepto se vuelve recursivo. Esto es, puedo usar un nombre más otro nombre, o un nombre más una inicial, o una inicial más un nombre, etcétera. Dicho de manera práctica, un nombre de pila puede ser *José* o *J.* o *José Luis* o *J. Luis* o *José L.* o *J. L.* . Al usar el concepto de recursividad una entidad se define como un nombre seguido de lo que se había utilizado como nombre con anterioridad; es decir, se pueden ir acumulando. Todos los nombres mencionados en el ejemplo caben dentro de esta definición. El símbolo <EOL> indica que lo del primer renglón es aparte de lo del segundo y no deben mezclarse.
- Hemos definido también que la dirección está formada por el nombre de una calle, seguido de un número exterior y, opcionalmente, un número interior, seguido de un nombre de colonia, en ese orden.
- Hemos definido que el código postal está formado por el nombre de una ciudad, seguido por el nombre de una entidad federativa y el número de un código postal, en ese orden.

- Hemos definido que un código postal está formado por cinco dígitos consecutivos.
- Hemos definido a un dígito como cualquiera de los guarismos del cero al 9.

Si bien no hemos definido una serie de validaciones y restricciones con respecto a estos posibles datos, ya que se trata de un ejemplo muy sencillo como puede observarse, la definición de un ente llamado dirección postal es posible, precisa y poco ambigua cuando usamos un metalenguaje; en este caso la notación BNF. Esto es más fácil que tratar de explicarlo en lenguaje corriente, ya que habría que estar haciendo muchas explicaciones sobre cada elemento, y además el resultado es una definición que tiene mayor formalidad. Además estas definiciones pueden ser interpretadas por un programa de computador. Puede hacerse entonces que ese programa escriba las categorías en sitios precisos dentro de una pantalla, hoja o etiqueta de papel: primero el nombre, luego la dirección y luego el código. Puede hacerse un listado sólo de los nombres de las personas y omitir selo de más; puede hacerse un listado que ordene los nombres alfabéticamente por apellidos, etcétera. Todas estas acciones serían muy difíciles de efectuar si el nombre y la dirección de una persona fuesen una sola entidad escrita de corrido, con lo cual una máquina se vería imposibilitada de distinguir cada una de esas partes.

Cabe hacer notar que esta notación de Backus-Naur es sólo un ejemplo de un metalenguaje sencillo para ilustrar conceptos básicos sobre este tópico. No se usa en la práctica para describir textos ya que carece de un buen número de elementos necesarios para realizar un correcto encodificado de ellos mismos. Para ello se han creado metalenguajes propios para ese efecto.

HTML (HYPER-TEXT MARKUP LANGUAGE)

Como ya se ha mencionado, a principios de los noventas se creó el *World Wide Web, WWW*, o simplemente *Web*, como un mecanismo para editar y acceder a información a nivel mundial con base en las tecnologías de Internet o red global de computadoras. En un inicio la información que viajaba en el internet era sólo de tipo texto, y las instrucciones debían ser tecleadas a modo de comandos. El advenimiento de la Web le agrega interfases gráficas a la distribución de información; es decir que una página de la web contiene ahora además de textos, imágenes (fotografías, dibujos, diagramas, etcétera), así como otros elementos de acceso más cómodos para el usuario: botones para seleccionar opciones, menús, etcétera. Agrega también un elemento muy importante: el hipervínculo; es decir una *"liga"* con otro texto, que puede estar en esa misma página o en cualquier otra, incluso en otra computadora, sistema o hasta otro país. Gracias al hipervínculo, el usuario puede, al posicionarse sobre un punto en la pantalla, dirigirse hacia otro conjunto de información de su interés sobre la Web. Posteriormente se agregaron

al medio sonidos e imágenes en movimiento, como animaciones o video, para hacer de la edición en la Web un ambiente realmente multimedios.

Precisamente para poder efectuar esta edición de información por medio de la Web se desarrolla el metalenguaje HTML que como ya se ha visto nace como un subconjunto o aplicación muy específica de SGML, del cual sólo se deseaban algunas de sus características que permitieran describir las partes de una página pensando sobre todo en cómo iban a verse sobre la pantalla. Lo principal era la distribución física de la información sobre la pantalla, es decir el *lay out*: dónde debían quedar los encabezados o letreros, su tipo de letra, tamaño y colores; el diseño y colores de los fondos; ubicación, color, tamaño, etcétera de párrafos sobre la pantalla; la ubicación, tamaño, etcétera, de las imágenes sobre la pantalla; los botones para seleccionar alguna opción de entre un menú; las ligas con otras páginas, etcétera.

Como puede verse HTML podía definir un cierto número de entidades en una página a la cual podía arribar un usuario, y brindarle acceso a la información. Pero adolece de un serio defecto: los textos en su interior son manejados solamente como eso, como "textos"; es decir, como un conjunto de palabras contiguas, sin ningún discernimiento de su contexto, su significado, suponderación, sus relaciones, etcétera. Por esta razón al multiplicarse el número de páginas en la red se hizo necesario un mecanismo que nos permitiera saber en dónde podría aparecer algo sobre un tópico de nuestro interés. Nacen así los "buscadores", con el fin de poder buscar y encontrar algo en la red. El problema es que los textos sólo están marcados como textos, cadenas de palabras sin ninguna ponderación sobre su contexto o sus partes. Las primeras búsquedas sólo hacen un barrido basándose en la "fuerza bruta" de la computadora que trata de identificar las palabras contenidas dentro de esos textos. Todos los que hemos navegado y utilizado la red sabemos el resultado de estos procesos: docenas, cientos y hasta miles de referencias inútiles llenas de "ruido" e irrelevantes para nuestro propósito. Quienes conocen algo de catalogación bibliográfica saben que es muy diferente buscar por ejemplo, a Octavio Paz como autor, como título o como tema en una obra. Peor aún si sólo buscamos por "paz" fuera de todo contexto, pues encontraremos a otras personas apellidadas o llamadas "Paz", pero también noticias sobre la paz, o la falta de ella, documentos, tratados, estudios, etcétera, que nos hablan de este estado social y que nada tienen que ver con nuestro Premio Nobel.

Peor aún fue cuando las páginas comenzaron a ser puerta de entrada para enormes repositorios de información, digamos un catálogo de biblioteca ¿cómo encontrar la información que está dentro de esa base de datos, pero que físicamente no está en la página Web? Las palabras para recuperar documentos usadas como llaves están asociadas a la página en sí misma, no a los bancos de datos ligados a ellas. Esa información está perdida para la recuperación.

Los buscadores tuvieron que irse sofisticando para poder contender con este problema y hoy varios mecanismos se han introducido con este fin, como los "metadatos", los "buscadores inteligentes" y algunos otros aditamentos como: frecuencia de la palabra en una página, proximidad o contigüidad entre ellas, etcétera. Bastante se

ha avanzado en este sentido, y debemos reconocer que en la actualidad hay páginas con mejores índices y buscadores realmente más adecuados para encontrar información **pertinente**. Pero el problema viene de origen, desde la manera en que la información es preparada y descrita en HTML. Este formato tiene algunas otras deficiencias, entre las cuales las más relevantes son que HTML:

- No es extensible; es decir, el conjunto de etiquetas es cerrado y nadie puede definir sus propias etiquetas para requerimientos específicos. Por ejemplo, en el mundo bibliográfico querríamos tener etiquetas tales como <autor>, <título> o <ISBN> en lugar de la etiqueta genérica <p> de párrafo.
- No permite representar las especificaciones de las estructuras de los datos, como se requiere en la creación y uso de bases de datos.
- No provee soporte para validar los datos.

HTML, por lo tanto, **no maneja los aspectos de contenido** y no puede ser utilizado, por ejemplo, para calificar el contenido de páginas Web. HTML sólo se enfoca a la presentación de la información. Más allá de esto cualquier cosa requiere de un considerable esfuerzo de programación adicional, ya sea en forma de *applets* o de aplicaciones o de programas.

No obstante lo anterior, sería un grave error menospreciar el HTML. Este estándar ha logrado resultados inusitados en la expansión de la divulgación documental electrónica en los últimos años. Gracias a él la Web ha llegado a ser lo que es hoy en día. HTML reforzó el hecho de que la plataforma de cómputo fuese irrelevante para el intercambio de información, y creó las bases del mecanismo de transporte para mover documentos a lo largo de distintas redes (lo que llegó a ser el protocolo HTTP, *Hyper-Text Transport Protocole),* y también el esquema de direccionamiento de documentos tanto locales como remotos (lo que llegaría a ser el direccionamiento URL *Universal Resource Locator*). Bastante se hizo con HTML a pesar de su estrechez y sus limitaciones, simplemente resulta que las necesidades de manejo documental siguen creciendo y empieza a verse limitado para contender con ellas. Como muchas otras grandes herramientas el HTML está siendo rebasado por el mismo dinamismo de la información de la red. El consorcio W3C ha considerado otras opciones que puedan contender con esas demandas y es por ello que SGML es vuelto a tomar en cuenta y nacen nuevos formatos como XML o XHTML. Sin embargo por un buen tiempo seguramente se seguirán viendo muchas aplicaciones desarrolladas en HTML. La aparición de XML y su uso seguramente se implantarán por medio de un cambio gradual de los desarrollos actuales y no mediante una sustitución inmediata.

Si bien los esfuerzos del grupo de trabajo del W3C se concentraron durante un tiempo en la definición del XML, como lo hemos ya mencionado, recientemente se liberaron las especificaciones de una primera versión de un metalenguaje denominado XHTML (eXtensible Hyper-Text Markup Language) compatible con XML, cuyas especificaciones han sido liberadas con anterioridad por el mismo consorcio [XHTML

1.0]. En lo personal no me queda muy claro por qué existen dos sucesores de HTML provenientes de la misma fuente. Como ya se ha explicado, HTML se veía cercado por una serie de limitaciones que eran resueltas con base en aplicaciones adicionales, y por ello un grupo de trabajo desarrolla un nuevo metalenguaje que, conteniendo al anterior (HTML), pueda contender con las necesidades actuales, y por esto se desarrolla y libera así la primera versión de XML; hasta ahí todo suena muy lógico. Por qué el mismo consorcio libera después una nueva versión de TML y la llama XTML y es compatible con ML no está claro y crea mucho mayor confusión en los desarrolladores al no saber éstos si sus nuevas aplicaciones provenientes de HML deberán hacerse en XHML o en ML. Pienso que el W3C quiso condescender con diversas facciones de grupos de desarrolladores y aceptó esta nueva versión del HTML convertida a XML con objeto de dar legusto a todos, pero me parece que la versión con más futuro será a la larga la denominada XML.

SGML (Standard Generalized Markup Language ISO 8879:1986)

Como ya se ha comentado, el SGML sirvió de base al HTML. El primero, mucho más completo que su sucesor, además de los aspectos propios de la presentación del documento puede contener una o más definiciones de tipos de documentos (DTD, o *"Document Type Definition"*), las cuales son descripciones formales de la sintaxis de los documentos que se le asignan a cada tipo de ellos que se desea definir. Por lo tanto, se requiere de una definición DTD para poder interpretar y verificar un documento SGML.

SGML tiene entre sus principales características ventajosas; primero, el ser un estándar no propietario, es decir, que no está atado a una marca, patente o compañía en particular, y que es apoyado por un gran número de proveedores de *software*. Por ello, un documento que cumpla con los estándares SGML tendrá una vida más larga que uno basado en un estándar propietario. En segundo lugar, los documentos codificados bajo SGML aún son bastante legibles para las personas y por tanto a la vez legibles para los programas de computadora. En tercer lugar, los documentos bajo SGML describen la estructura de los datos y su semántica, y no sólo la manera en que van a ser presentados en la pantalla.

Entre sus desventajas podemos mencionar: primero, el hecho de que es muy generalizado, que incluye especificaciones particulares para todo tipo de documentos y que se vuelve muy complejo: sus especificaciones se extienden por más de 500 páginas. En segundo lugar, muchas de esas especificaciones son irrelevantes para el uso del documento en la Web, lo cual las hace superfluas cuando ése es el uso que se pretende darle al documento. Dado que hay muchas opciones dentro de éste, la interoperabilidad entre diversas empresas se reduce sensiblemente.

Por lo anterior, los principales usos de SGML se han dado en ambientes "cerrados" donde es más o menos fácil controlar la estandarización de documentos, tales

como los sectores militares, de inteligencia, de manufactura de aeronaves, empresas de publicación, grandes sistemas de archivo, etcétera. [SGML ISO 8879, 1986].

XML (eXtensible Markup Language)

En los anteriores capítulos hemos visto ya que las ventajas y desventajas de HTML y SGML se contraponen: HTML es práctico y esbelto, pero ha sido rebasado por las necesidades del momento en la Web, y GML, siendo mucho más completo tiene muchos elementos imprácticos para su uso en la red. Hemos visto también que el W3C, tratando de contender con esta problemática, favoreció el desarrollo de un nuevo estándar que trata se de tomar "lo mejor de ambos mundos" y es así como nace el nuevo estándar XML, que pretende combinar las mayores capacidades descriptivas y semánticas del SGML con lo práctico, sencillo y universal del HTML. Podría afirmarse que se trata de **"la versión corregida y aumentada de HTML" o "un SGML aerodinámico para la Web"**.

En efecto, XML presenta una serie de ventajas prácticas que lo hacen ideal para ir sustituyendo al HTML en la web, he las aquí:

- Sus especificaciones vienen contenidas en sólo 26 páginas.
- Los programas analizadores sintácticos de XML, que se construyen en una computadora dada, no requieren del mencionado descriptor del tipo de documento (DTD) para poder separar un documento en sus componentes.
- No permite ninguna desviación de la sintaxis estándar. Esto quiere decir que todos los documentos codificados en XML pueden ser editados, guardados y enviados sin importar el programa analizador de XML del receptor.
- Los documentos XML pueden proveer *hojas de estilo* que le permiten a los navegadores (Netscape, Explorer, etcétera) convertir los documentos para presentarlos en HTML en la pantalla.
- Los analizadores XML pueden analizar incluso documentos HTML bien formados. Esto permite que la transición de documentos ya existentes en HTML en una empresa que esté evolucionando hacia XML pueda hacerse de manera gradual y programada.

Como ya se ha establecido, XML es un metalenguaje que permite establecer un conjunto de reglas para definir una sintaxis específica, la cual será aplicada solamente en un escenario preestablecido. Los documentos que pueden especificarse entonces bajo XML pueden ser de muy diversas naturalezas, por ejemplo las propias del mundo de las finanzas, al especificar datos de acciones, cotizaciones, tipos de cambio de divisas, etcétera; o **los datos de tipo bibliográfico contenidos en un documento dado**; o bien datos de compuestos químicos dentro de una fórmula; o planos y datos sobre la construcción de un avión, etcétera.

Se han presentado una serie de postulados acerca de las principales diferencias entre el estándar HTML y el XML. Tal vez la mejor manera de comprender esto plenamente

sea con un pequeño ejemplo que lo ilustre. Tomemos un mínimo trozo del guión dramático *Don Juan Tenorio*, tal como aparece el texto impreso en un libro (sin lenguaje de marcado electrónico):

ACTO I
Escena 1. Don Juan, con antifaz, sentado a una mesa escribiendo; Ciutti y Buttarelli a un lado esperando. Al levantarse el telón se ven pasar por la puerta del fondo máscaras, estudiantes y pueblo con hachones, música, etcétera.
Don Juan: ¡Cuán gritan esos malditos!
¡Pero mal rayo me parta
Si en concluyendo la carta
No pagan caro sus gritos!
Buttarelli: *A Ciutti*
Buen carnaval
Ciutti: *A Buttarelli*
Buen agosto
Para rellenar la arquilla.

Para leer e interpretar este texto de Zorrilla, una persona no requiere marcado adicional. Es capaz de reconocer el significado de cada una de las partes que lo forman basándose en su experiencia y el simple *"marcado"* tipográfico.

La versión HTML de este texto se vería así:

```
<H1> ACTO I </H1>
<P><I>Escena 1. Don Juan, con antifaz, sentado a una mesa escribiendo; Ciutti y
Buttarelli a un lado esperando. Al levantarse el telón se ven pasar por la puerta
del fondo máscaras, estudiantes y pueblo con hachones, música, etcétera.</I></P>
                <P><B>Don Juan:</B> ¡Cuán gritan esos malditos!
                ¡Pero mal rayo me parta
                Si en concluyendo la carta
                No pagan caro sus gritos! </P>
<P><B>Buttarelli:</B>       <I>A Ciutti</I>
                Buen carnaval </P>
<P><B>Ciutti:</B>    <I>A Buttarelli</I>
                Buen agosto
                Para rellenar la arquilla.</P>
```

La nomenclatura de marcado HTML es la siguiente:

<> encierra el principio de una etiqueta de marcado.
</> encierra el fin de una etiqueta de marcado. Siempre van por pares: <> comienza marcado; </> termina marcado para esa etiqueta.

H significa *"Header"* o encabezado. Puede haber más de uno y por tanto se numeran: H1 es el "Header 1". <H1> marca de comienzo del encabezado 1. </H1> marca del fin del encabezado 1.

P significa *"Paragraph"* o párrafo. <P> y </P> indican el principio y el fin de un párrafo.

I significa "Itálicas". "B" significa *"Bold"*, negritas. Puestos en pares para indicar dónde comienza y termina esa característica del texto, con ayuda de la diagonal "/".

La versión XML de este mismo texto podría ser:

```
<ACT> <TITLE>ACTO I</TITLE>
<SCENE><TITLE>Escena 1. Don Juan, con antifaz, sentado a una mesa escribiendo; Ciutti y Buttarelli a un lado esperando. Al levantarse el telón se ven pasar por la puerta del fondo máscaras, estudiantes y pueblo con hachones, música, etcétera.</TITLE>
<SPEECH>
<SPEAKER>          Don Juan:</SPEAKER>
<LINE>
                ¡Cuán gritan esos malditos!
                ¡Pero mal rayo me parta
                Si en concluyendo la carta
                No pagan caro sus gritos!
</LINE>
</SPEECH>
<SPEECH>
<SPEAKER>Buttarelli:</SPEAKER>
                <LISTENER>A Ciutti</LISTENER>
                <LINE>Buen carnaval</LINE>
</SPEECH>
<SPEECH>
<SPEAKER>Ciutti:</SPEAKER>
                <LISTENER>A Buttarelli</LISTENER>
                <LINE>Buen agosto
                para rellenarla arquilla.</LINE>
</SPEECH>
.....
```

Las terminaciones de los marcadores </SCENE> y </ACT> estarán hasta donde se encuentre el final de la escena y el final del acto, por tanto no se escriben aquí.

Los marcadores <> y </> (abrir y cerrar etiqueta) se manejan igual que en HTML y por pares.

Si se observan estos ejemplos marcados se puede notar que en el primer caso, con HTML, los marcados nos indican encabezados, párrafos, itálicas y negritas; es decir, **indicaciones para el despliegue del texto, pero nada sobre las partes o el contexto**

que van componiendo la obra. En el segundo ejemplo, el de XML, puede observarse que se ha marcado, además del encabezado o título, dónde empieza el acto I; dónde empieza la escena; dónde empieza y termina el título; dónde comienza y termina cada diálogo (*speech*); quién es el actor del diálogo y quién es el que escucha, y cuál es la línea del diálogo que le pertenece a cada quien. De esta forma, el documento ha quedado marcado con una serie mucho más completa de elementos, la cual es susceptible de ser analizada no sólo por los seres humanos sino también por una computadora.

Supongamos que queremos que la computadora reproduzca por medio de un programa sintetizador de voz los diálogos a través de las bocinas de la misma; por supuesto con voces apropiadas para cada personaje. Con el primer marcado, el HTML, sería muy difícil lograrlo; con el segundo, la máquina puede, al ir "leyendo" el texto, identificar fácilmente quién es el actor que habla <SPEAKER>, y utilizar siempre la misma voz para ese personaje, y reproducir sólo el texto que se encuentra en <LINE>, ignorando para el parlamento las indicaciones de escena, título, nombre del actor, quién es su interlocutor, etcétera.

A propósito hemos denominado las etiquetas en inglés: *title, act, speech, listener, line*, etcétera. Podríamos haberlas denominado título, acto, diálogo, escucha, línea, etcétera. Sería válido y funcionaría en nuestro ámbito. Pero ¿Qué pasaría si ese diálogo quiere ser puesto en la Internet para su uso universal? ¿Cómo sabrán las máquinas de otras partes del mundo que <diálogo> es eso, un diálogo? Si bien no hay un estándar establecido en cómo debe llamarse a una etiqueta, es muy probable que en poco tiempo, los dramaturgos que quieran poner sus textos en esta forma se inclinen por un nombre universal para la marca, y lo más probable es que sea un nombre en inglés y que, por convención, todo el mundo acepte en poco tiempo que "<speech>", por ejemplo, es siempre un diálogo en una obra de teatro.

Otro ejemplo de aplicación de este marcado es que una computadora, bajo pedido, puede hacer fácilmente una lista de los personajes de la obra, <SPEAKER> y ponerla en su directorio de palabras para recuperación, con objeto de que un usuario en la red busque y encuentre a tal o cual personaje que aparece en una obra. Hacer esto con el primer marcado sería sumamente difícil.

Cabe hacer notar aquí que en ambos ejemplos no hemos utilizado todos los elementos posibles del marcado de cada metalenguaje. Ni con mucho hemos agotado todas las etiquetas utilizables. Por supuesto, en XML pueden agregarse marcas para tipografía, pero nosotros sólo hemos seleccionado unas cuantas, simplificando el ejemplo, para ilustrar el concepto. No es el propósito de este documento presentar todos los elementos de marcado de cada formato, porque esto sería inmenso, pero sí existen textos que hacen esto. La idea es resaltar las características descritas a lo largo de este texto.

Resumiendo, podríamos afirmar que XML es:

Simple - La especificación completa mide menos de 30 cuartillas. XML ha sido diseñado para facilitar aún más la escritura de programas con respecto a HTML o SGML.

Extensible - Cada quien puede inventar sus propias etiquetas para marcar cualquier tipo de documento, y ser éstas compartidas. De hecho, XML es un metalenguaje que le permite al usuario definir su propio lenguaje de marcado.

Un estándar abierto - XML es SGML. Ello significa que no es necesario saber programar; existen muchas herramientas eficientes que permiten ya sea crearlo, manejarlo o implantarlo en una computadora y distribuirlo.

Eficiente - XML tiene entes interconstruidos para reutilizar fragmentos de documentos, así, estos sólo tienen que ser transmitidos una vez.

Basado en la experiencia - XML ha sido diseñado por personas que tienen amplia experiencia en los lenguajes de marcado y han capitalizado las enseñanzas que el uso de ellos les ha suministrado a lo largo de los años.

Consensuado - El diseño de XML incluye los puntos de vista de los organismos coordinadores de HTML y SGML, así como los de personas que han desarrollado importantes aplicaciones con estos estándares.

Libre - Nadie tiene la propiedad o patente de XML, ni podrá tenerla, ya que tanto SGML como XML han sido definidos como estándares internacionales. Por lo mismo su uso o desarrollo no implica el pago de ninguna regalía.

Internacional - XML tiene interconstruido un soporte para textos en prácticamente todos los alfabetos del mundo, incluyendo técnicas para consignar el lenguaje y/o código del alfabeto utilizado.

Listo para ser usado - Los *"browsers"* o *navegadores* del web, en sus últimas versiones, son capaces de leer especificaciones XML. Los hipervínculos, textos y multimedios pueden ser vistos tal como se hace ahora con un documento HTML.

Manejable - XML incluye métodos para declarar y reforzar las estructuras documentales usadas actualmente, como las de bases de datos.

Validable - XML tiene técnicas que permiten la validación de los documentos involucrados, así que uno puede estar seguro de que los documentos registrados con él son creados correctamente.

ANÁLISIS DE LA PROBLEMÁTICA

Para poder proceder a un análisis crítico de la situación es necesario primero acotar el entorno en el cual deseamos realizarlo.

Sin duda una de las características del establecimiento de colecciones y bibliotecas digitales es que éstas no se limitan sólo a materiales de referencia, sino, cada vez más, a textos completos. Las bibliotecas digitales tratan ahora de establecer los mejores mecanismos para almacenar estos textos completos electrónicos de tal forma que sean totalmente recuperables y explotables por las comunidades académicas, más allá de hacer una simple búsqueda de palabras en el texto y operadores booleanos. Ha quedado demostrado que XML tiene un gran número de ventajas nativas para este

propósito, y que por ello mismo será el lenguaje de marcado más utilizado en esta década hasta que sea sustituido por algo mejor. No obstante, su misma universalidad y extensibilidad, y las mayores ventajas de XML, introducen una serie de problemas en el entorno en que éste puede ser utilizado.

En efecto, debemos empezar a acotar el problema en nuestro medio; como se ha mencionado XML puede ser utilizado para describir toda clase de documentos: desde los rollos del Mar Muerto, pasando por un expediente médico o un libro, hasta las especificaciones de un avión supersónico. Puede ser usado entonces en los sectores educativo, de la construcción, médico, gubernamental, etcétera. Cada sector puede y debería empezar a sentar las bases para definir sus documentos más relevantes. De hecho, puede existir más de una definición para documentos de un mismo sector o tipo, lo que haría muy difícil su manejo por parte de usuarios distintos a aquellos para los cuales se haya diseñado la información.

Por este motivo en la actualidad se están definiendo descriptores de tipo de documento (DTDs) por grupos sectoriales con intereses afines, de forma que están surgiendo estándares avalados por organismos que garantizan que cualquier usuario que los adopte como suyos, trabaje con las mismas etiquetas e idénticas normatividades, como se hace con el actual HTML. Como ejemplos de esto tenemos CML, *Chemical MarkupLanguage* para el sector químico, MathML, *MathematicalMarkupLanguage* para definir datos matemáticos, SMIL, *SynchronizedMultimediaIntegrationLanguage*, para definir presentaciones en recursos multimedios, etcétera. A estos tipos de variantes de documentos se les conoce como *clases* y es un término muy importante.

De hecho para un mismo tipo de documento pueden existir diversos enfoques y necesidades. Tomemos el caso del documento "libro". La definición de este documento obedecerá a los intereses particulares de determinado sector. Por ejemplo, para alguien que comercializa libros en la Web, su definición de documento sólo incluiría elementos mínimos de identificación: autor, título, año, editorial, ISBN; pero también datos tales como precio, descuento, peso, costo de envío, disponibilidad y/o tiempo de entrega, condiciones de pago, etcétera. Poco que ver con el ambiente de una biblioteca, en donde faltan muchos otros datos con fines de referencia de los documentos, mientras que algunos otros de nuestro ejemplo salen sobrando. Si el documento por definirse es un texto completo, deben agregarse mucho más datos. Por ello las definiciones hechas para un sector serían inútiles para otro.

En el medio académico, y en particular el de las bibliotecas, es necesario entonces comenzar a establecer las definiciones tipo para cada clase de documento, con objeto de satisfacer cabalmente las necesidades de consulta por parte de sus comunidades de usuarios. Cabe recordar que existen además distintos tipos de bibliotecas, y por ende, deberá haber ciertos "matices" en estas definiciones. Estos estándares deberían, por supuesto, estar acordes con nuestra realidad mexicana y con las características de nuestra producción editorial, entorno histórico y cultural, idioma, etcétera.

Debemos reflexionar entonces si queremos sentarnos a esperar que estas definiciones sean hechas por otros entornos extranjeros, o por sectores ajenos a la academia y a

la biblioteca, o por personal no profesional en el registro documental; o si es el medio bibliotecario académico el que debe comenzar a sentar las bases para establecer estándares en estas definiciones, como ya lo han comenzado a hacer otros sectores en otras partes del mundo. **El sector bibliotecario es el que debería hacer las especificaciones de los documentos para el medio académico**, y sería ideal que fuera el sector bibliotecario mexicano el que lo hiciera para el medio académico mexicano.

Hemos visto ya el despertar y el auge que están tomando las bibliotecas digitales en todo el orbe, y el que comienzan a tener en nuestro país. Empezamos a ver ya algunas colecciones de documentos digitales. Sin embargo, cabe observar que casi todas estas colecciones están presentadas en formato HTML y que casi ninguna de estas colecciones de bibliotecas digitales tiene ya sus definiciones XML, aun que hay que destacar que algunas ya lo están considerando y preparándose para ello. Se venden o distribuyen ya en el medio algunos programas o aplicaciones que se anuncian como "generadores de bibliotecas digitales", tanto de origen nacional como extranjero. Empiezan a surgir ya las primeras bibliotecas digitales con especificaciones XML. La pregunta crucial es ¿quién crea esas definiciones para los tipos de documentos que manejan esas bibliotecas? Un examen más detallado nos enseñará que por lo general no han sido hechas por personal profesional en registro documental, y por lo tanto cuando mucho representan el punto de vista de un sólo profesional o sector. Las "etiquetas" de marcado muchas veces son sólo réplicas de las establecidas por MARC o AACR2, que son buenas para material de referencia pero no para textos completos.

Cabe resaltar aquí que estas deficiencias no se hacen notar de un modo peyorativo alguno. Las personas u organizaciones que están detrás de ello han hecho su mejor esfuerzo con los elementos disponibles hasta este momento y sin ningún estándar prestablecido y por territorios no explorados. Simplemente deseamos resaltar el hecho y el riesgo que se corre de crear una Babel al rededor del tema, hasta que los mejores estándares desarrollados se vayan filtrando en el medio y la situación se estabilice. Pero ello puede tomar mucho tiempo y costar caro, además de consumir recursos que en nuestro país no sobran y deben ser utilizados racionalmente para el mejor desarrollo de nuestras bibliotecas y colecciones digitales y, por ende, de las comunidades académicas a las que éstas sirven.

Es altamente necesario, entonces, comenzar a establecer los estándares mínimos que estas definiciones deben contener para cada tipo de documento orientado hacia el sector académico en general y bibliotecario en particular, considerando no sólo las características de referencia sino los textos completos, y enfocándolo a nuestro medio mexicano, a nuestras bibliotecas y a nuestro entorno histórico, social, económico, etcétera. Estas definiciones deben ser efectuadas por personal experto, profesional y multidisciplinario en tareas de registro, recuperación y distribución documental, con el fin de que sean avaladas por las bibliotecas digitales que se están creando y adoptadas por ellas como suyas. La idea es crear un ambiente de homogeneización y calidad en los registros y las colecciones que se formen.

No se trata de ponerle una camisa de fuerza a las especificaciones de los documentos, sino de crear una definición de calidad sobresaliente como base que pueda ser aceptada con confianza por las bibliotecas digitales de nuestro medio, para que a partir de ella cada biblioteca pueda seguir construyendo sus propias especificaciones.

¿Qué es lo que habría que definir entonces?

ALCANCES DEL ESTUDIO

Típicamente un documento XML tiene dos partes: una definición del lenguaje que se usa, y el contenido del documento en sí mismo. La definición puede ser escrita usando una DTD (definición o descripción del tipo de documento), que es una sintaxis para describir en forma de HTML. Esta DTD puede estar inmersa en el propio documento o ser externa a él. Al ente descrito se le llama **objeto**. Esta técnica ha sido usada hasta este momento con gran intensidad y éxito, dada la reciente liberación del estándar XML. De inicio, sería tonto ja que el proyecto debería tender a desarrollar las especificaciones detalladas mencionadas anteriormente en forma de DTDs que puedan ser distribuidas para lograr el propósito establecido.

Sin embargo en los últimos meses se ha venido desarrollando una forma más eficaz para definir documentos bajo el estándar XML, conocida como **esquemas de tipo de elemento** o simplemente **esquemas**, que definen las características de las clases de objetos. Estos esquemas pueden definirse como un DTD que permite además su propia ampliación mediante un metalenguaje propio creado al efecto, que define a su vez al DTD. Dicho de otra forma, el esquema representa el metadato para un documento XML asociado, y describe qué puede y qué no puede ser incluido en un documento XML [Campbell, 2003]. Con esta técnica pueden desarrollarse vocabularios especiales para definir y documentar clases de objetos. Esta técnica promete mayores y mejores alcances para definir documentos y se perfila como más interesante como proyecto de investigación de frontera. El presente trabajo pretende explicar lo que es un esquema XML y descubrir las ventajas más allá de un DTD. [Young, 2002]

Vamos a tratar de ir ilustrando esto, para empezar ¿cómo se ve un esquema de tipo de elemento? aparentemente no difiere mucho de un descriptor DTD. Lo mejor será ilustrarlo con un pequeño ejemplo. Todas las declaraciones de un esquema están contenidas en un **elemento de esquema** como en el siguiente ejemplo:

ESQUEMA DE TIPO DE ELEMENTO

```
<?XML version='1.0' ?>
<s:schema id='EjemploEsquema'>
<!– el esquema va aquí. –>
.....
.....
</s:schema>
```

Declaración del Tipo de Elemento

El núcleo de un esquema de dato XML es la declaración de **Tipo de Elemento**, la cual define una clase de un objeto. El atributo **id** tiene la doble característica de identificar la definición y de nombrar una clase específica. Los descriptores básicos de nomenclatura ya están definidos en XML y están en inglés. A esto se le denomina el dominio **namespaces** en XML y es una tabla ya definida, si bien en constante actualización. No deben confundirse los descriptores básicos de nomenclatura (*string, integer, maxxocur*, etcétera) ya establecidos, con los nombres posibles de las etiquetas. Ésa es precisamente la ventaja de XML sobre HTML o MARC: los descriptores **no** están establecidos y uno puede bautizar los como uno considere pertinente. Si bien podríamos bautizar a todos y cada uno de los elementos en los descriptores en castellano: autor, título, tema, etcétera, en una sana defensa de nuestra lengua, ello no será conveniente en la práctica, porque esta definición que daría sólo para uso local en los países de habla hispana, y como lo que se desea es que estas definiciones sean de índole internacional, tal vez lo mejor sea hacerlo en inglés. No obstante lo anterior, es posible establecer "alias" de los nombres, y estos alias pueden ser las nomenclaturas en castellano, con lo cual pueden utilizarse ambas nomenclaturas agregando unas cuantas definiciones más y cuidando que existan simultáneamente. En los ejemplos subsecuentes utilizaré descriptores en español para resaltar la facilidad con que uno puede denominarlos.

Para entender cómo se construye un esquema debemos entender primero cómo se establecen las definiciones de los elementos. Hagamos unos ejercicios sencillos previos de cómo se define en XML. Utilicemos para el ejemplo la definición de un elemento "autor" dentro de un documento "libro":

```
<ElementType id="autor"/>
```

Dentro del tipo del elemento, podemos incluir un subelemento **descripción** (*description*), para que las personas puedan descifrar cuál es el contenido de ese elemento. De hecho podemos agregar múltiples subelementos a un elemento con objeto de dejarlo perfectamente definido. [Thompson, 1997]

```
<ElementType id="autor"/>
        <description>La persona o institución que escribió la obra.</description>
</ElementType>
```

Propiedades y Contenidos

Dentro del ElementType podemos crear subelementos que definan las características de lo ahí contenido.

```
<ElementType id="autor">
        <string/>
</ElementType>
<ElementType id="libro">
        <ElementType="#autor" occurs="ONEORMORE"/>
</ElementType>
```

Este ejemplo define dos elementos, "autor" y "libro". Indica que un elemento "libro" puede tener uno o varios elementos "autor". El elemento autor puede contener una cadena (*string*) de caracteres. Dentro de un tipo de elemento pueden estar contenidos varios subelementos (*element, group, any, empty, string*, etcétera). Éstos indican cuáles pueden ser incluidos en ese tipo de elemento e incluso su secuencia. Los tipos de elementos tienen también un atributo **ocurre** (*occurs*), el cual puede tener cuatro valores: requerido *"required"*, opcional *"optional"*, cero o más *"zeroormore"*, y uno o más *"oneormore"*.

Siguiendo estas reglas, podemos continuar construyendo esta descripción:

```
<ElementType="#titulo" occurs="OPTIONAL"/>
```

Podemos agregar también unos subelementos en "grupo"; digamos "prólogo" e "introducción", los cuales pueden venir en el documento libro. Cabe hacer aquí la aclaración de que estamos definiendo un documento para ser ingresado en texto completo, y no sólo los elementos de su ficha catalográfica.

Un registro más elaborado (sin pretender ser exhaustivo) y sólo a manera de ilustración, podría verse de esta forma:

```
<ElementTypeid="libro">
        <ElementType="#autor" occurs="OPTIONAL"/>
        <ElementType="#titulo" occurs="ONEORMORE"/>
        <ElementType="#fecha-de-copyright" occurs="OPTIONAL"/>
        <groupocccurs="OPTIONAL">
                <elementType="#prefacio"/>
                <elementType="#introduccion"/>
        </group>
        <ElementType="#ISBN" occurs="OPTIONAL"/>
        <ElementType="#texto-del-libro" occurs="ONEORMORE"/>
        "ElementType="#para-edades" occurs="OPTIONAL"/>
</ElementType>
```

Un documento marcado acorde con esta estructura podría verse así:

```
<libro>
        <autor> Miguel de Cervantes Saavedra </autor>
        <titulo> El IngeniosoHidalgo Don Quijote de la Mancha
        </titulo>
        <prefacio>Aquí iría un prólogo escrito.....</prefacio>
        <introduccion> Aquí iría una introducción.... </introduccion>
        <isbn> 123-456-789-0 </isbn>
        <texto-del-libro> En un lugar de la Mancha, de cuyo nombre no quisiera
          acordarme.... (sigue todo el texto del libro)
        </texto-del-libro>
        <para-edades> todo público</para-edades>
</libro>
```

Uno de los aspectos de mayor importancia en la definición del esquema, consiste en que, como ya habíamos mencionado, pueden establecerse "alias" de los descriptores. Un "alias" consiste, como su nombre lo indica, en un segundo elemento de identificación de un nombre de elemento. **Su importancia radica en que estos alias pueden ser usados para tener una equivalencia en castellano de cualquier nombre definido previa o posteriormente.** El descriptor se llama *elementTypeEquivalent*.

```
<elementTypeEquivalent id="libro" type="#book"/>
<elementTypeEquivalent id="autor" type="#author"/>
<elementTypeEquivalent id="titulo"type="#title"/>
<elementTypeEquivalent id="personaje"="#personaje"/>
```

...etcétera.

De esta forma, dentro de la definición del documento, "libro" y "book" **se vuelven sinónimos y pueden ser usados indistintamente dentro de la definición**. Así, con este tipo de elementos, podemos mantener la internacionalidad de la definición y seguir manejando los términos en nuestra lengua.

Ahora bien, este fue sólo un pequeño ejemplo relacionado con los atributos de las primeras versiones de XML para facilitar su comprensión, pero su uso puede irse expandiendo y sofisticando hasta dimensiones inimaginadas. Hemos definido un elemento "autor" formado de una sola cadena de caracteres, pero ello no tiene por qué limitarse a eso. Podemos definir autor como una serie de elementos, digamos, nombre, apellidos, título o cargo, fechas de nacimiento y muerte, seudónimos, etcétera. Suena más parecido a lo que estamos acostumbrados a ver en las reglas angloamericanas de catalogación AACR2. Igualmente podemos subdividir título y subtítulo, y pie de imprenta en ciudad, editorial y año, etcétera.

Pero la definición no se queda en los elementos de la ficha catalográfica. Como se ha visto podemos describir el texto completo de un libro. Podemos subdividirlo en capítulos, secciones, apartados, etcétera. Podemos señalar notas a pie de página, citas bibliográficas, palabras a ser referidas en un glosario, direcciones de otras páginas web, etcétera. Podemos marcar personajes, escenas, lugares, diálogos expresados o pensamientos del personaje, etcétera, como ya lo hemos mostrado en otros apartados previos.

Podemos también, agregando algunas técnicas llamadas API (*Application ProgrammingInterface*), marcar diferentes valores para algún elemento; es decir, podemos predeterminar si ese elemento es una cadena de caracteres, un número entero o fraccionario, e incluso contar con rangos posibles; una fecha en algún formato dado (ISO proporciona varios estándares), un booleano, un elemento que debe existir de acuerdo con una tabla o catálogo preestablecidos, etcétera.

CONSTRUYENDO UN PRIMER ESQUEMA

Una vez que hemos hecho una pequeña presentación de un esquema podemos pasar a construir un esquema verídico tal como puede ser construido en la vida real. Para poder iniciar esta construcción partiré de un ejemplo real de un descriptor de tipo de documento (DTD) para un libro y de ahí continuaré con la construcción de un esquema. Los nombres de los atributos de los *tags* o etiquetas están tomados del XML *namespaces* para fines de uso ya normalizado, de acuerdo con lo especificado por XML. Para iniciar el ejemplo utilizaré la obra del Tenorio ya presentada con anterioridad según el orden de Van der Vlist [Van der Vlist, 2001]; de acuerdo con un DTD ya establecido podría verse así:

```
<?xml version="1.0" encoding="UTF-7,5"?>
<libro>
<titulo> Don Juan Tenorio </titulo>
<autor>José Zorrilla </autor>
<isbn="8420639028">
<serie="El libro de Bolsillo. Literatura">
<personaje>
        <nombre>Don Juan Tenorio</nombre>
        <representa-a>el galán de la obra </representa-a>
        <desde >1844</desde >
        <cualificacion>
                galán extrovertido calavera y pendenciero
        </cualificacion>
</personaje>
<personaje >
        <nombre>Doña Inés de Ulloa </nombre>
                <representa-a > la dama joven de la obra pretendida por Don
        Juan</representa-a>
        <desde>1844</desde>
        <cualificacion>
                cándida y angelical monja pretendida por Don Juan
        </cualificacion>
</personaje>
<personaje>
        <nombre> Marcos Ciutti</nombre>
        <representa-a> El criado de Don Juan</representa-a>
        <desde>1844</desde>
        <cualificacion> hábil, pícaro, diligente y fiel sirviente de Don Juan
        </cualificacion>
</personaje>
</libro >
```

Para escribir un esquema que describa los documentos marcados en esta forma, simplemente seguiremos su estructura y elementos conforme los vamos encontrando; para empezar, abrimos el elemento *xs:schema*:

```
<?xml version="1.0" encoding="UTF-7,5"?>
<xs:schema
        xmlns:xs="http:// www.unam.mx/xmlimaginario/XMLSchema">
.../...
</xs:schema>
```

La dirección http indicada es donde debe residir el esquema; en este caso es ficticia. Sirve para abrir nuestro esquema y también guarda la definición del *namespace* destino y varias opciones que se asignan por omisión, algunas de las cuales serán explicadas más adelante.

Para cuadrar la etiqueta inicial para el elemento "libro", definimos un elemento con ese nombre. Este elemento tiene atributos e "hijos" no-texto, por lo que lo consideramos como de tipo complejo *complexType*, dado que los otros *dataTypes* como *simpleType* están reservados para tipos de datos que contienen sólo valores predeterminados y no elementos o sub-nodos de atributo. La lista de "hijos" del elemento "libro" es descrita por un elemento de secuencia *sequence*:

```
<xs:element name="libro">
        <xs:complexType>
                <xs:sequence>
                .../...
                </xs:sequence>
                .../...
        </xs:complexType>
</xs:element>
```

La secuencia es una "guía" que define una secuencia ordenada de sub elementos. Veremos otras guías, *choice* and *all* en las siguientes secciones.

Podemos definir ahora los elementos de autor y título como tipos simples, no tienen atributos o hijos no-texto y pueden ser descritos directamente como elementos. El tipo *xs:string* (cadena de caracteres) los define de esa forma:

```
        <xs:element name="titulo" type="xs:string"/>
        <xs:element name="autor" type="xs:string"/>
```

Ahora establecemos el elemento "personaje", el cual es de tipo complejo y describe a los diferentes personajes de la obra. Nótese cómo se define su cardinalidad, es decir, el número de veces que puede ocurrir:

```
<xs:element name="personaje" minOccurs="0"
 maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
.../...
</xs:sequence>
</xs:complexType>
</xs:element>
```

Especificamos la lista de todos sus "*hijos*" de la misma forma:

```
<xs:element name="nombre" type="xs:string"/>
<xs:element name="representa-a" type="xs:string"
        minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="desde" type="xs:date"/>
<xs:element name="cualificacion" type="xs:string"/>
```

Terminamos su descripción cerrando el *complexType*, y los elementos *element* y *sequence*. Podemos ahora declarar otros atributos de los elementos del documento:

```
<xs:attribute name="isbn" type ="xs:string"/>
<xs:attribute name="serie" type ="xs:string"/>
```

Y se cierran los elementos restantes.

Este diseño, conocido en el medio como de tipo *matriushka* por las muñecas rusas de madera que se anidan una dentro de la otra, sigue de cerca la estructura del documento muestra. Una de las características clave de tal diseño es la de definir cada elemento y atributo dentro de su contexto permitiendo múltiples ocurrencias del mismo elemento para obtener diferentes definiciones.

Lista completa del primer ejemplo de esquema:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
   xmlns:xs="http://www.unam.mx/xmlimaginario/XMLSchema">
        <xs:element name="libro">
           <xs:complexType>
              <xs:sequence>
                 <xs:element name="titulo" type="xs:string"/>
                 <xs:element name="autor" type="xs:string"/>
                 <xs:element name="personaje" minOccurs="0"
                    maxOccurs="unbounded">
                 <xs:complexType>
                 <xs:sequence>
```

```
                    <xs:element name="nombre" type="xs:string"/>
                    <xs:element name="representa-a" type="xs:string"
                        minOccurs="0"
                        maxOccurs="unbounded"/>
                    <xs:element name="desde" type="xs:date"/>
                    <xs:element name="cualificacion"
                        type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
</xs:element>
</xs:schema>
```

Nótese cómo el primer documento descrito con marcadores XML corresponde a un documento, la obra del Teno rio de Zorrilla en particular. Este último listado corresponde al esquema de ese documento; es decir, con este esquema podemos describir todos los documentos u obras que hayan sido marcadas con el estilo del documento presentado. Podemos entonces procesar los documentos descritos en esa forma, pero si anexamos el esquema a los documentos los sistemas pueden saber cómo fueron construidos esos documentos. [Morrison. 2002]

DIVIDIENDO EL ESQUEMA

Si bien la estructura presentada anteriormente es muy simple, puede llevar a muchos anidamientos de las definiciones inmersas, haciéndola muy difícil de leer y difícil de mantener cuando los documentos son muy complejos. Tiene también la desventaja de ser muy diferente a una estructura de un DTD, lo cual es un obstáculo tanto para las personas como para los programas que deseen transformar un DTD en un esquema XML.

Un segundo diseño que presento está basado en una "tabla plana" de todos los elementos disponibles en un documento, y para cada uno de ellos, listas de los elementos "hijos" y sus atributos. Este efecto se logra a través del uso de referencias hacia los elementos y definiciones de atributos que requieren estar dentro del ámbito del referencista, y se ven en forma de una tabla plana como ya mencionamos.

```
<?xml version="1.0" encoding="UTF-7,5"?>
<xs:schema xmlns:xs=
    "http:// www.unam.mx/xmlimaginario/XMLSchema">
<!—definición de los elementos de tipo simple —>
<xs:element name="autor" type="xs:string"/>
```

```
<xs:element name="titulo" type="xs:string"/>
<xs:element name="nombre" type="xs:string"/>
<xs:element name="representa-a" type="xs:string"/>
<xs:element name="desde" type="xs:date"/>
<xs:element name="cualificacion" type="xs:string"/>
<!—definición de atributos—>
<xs:attribute name="isbn" type="xs:string"/>
<!—definición de elementos de tipo complejo —>
<xs:element name="personaje">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="nombre"/>
            <xs:element ref="representa-a" minOccurs="0"
                maxOccurs="unbounded"/>
            <xs:element ref="desde"/>
            <xs:element ref="cualificacion"/>
            <!—Los elementos de tipo simple son referenciadosusando el atributo
                "ref" —>
            <!—La definición de la cardinalidad se hace cuando el elemento es re
                ferenciado—>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="libro">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="titulo"/>
                <xs:element ref="autor"/>
                <xs:element ref="personaje"minOccurs="0"
                    maxOccurs="unbounded"/>
                </xs:sequence>
            <xs:attribute ref="isbn"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Usar una referencia hacia un elemento o atributo es el equivalente de "clonar" a un objeto. El elemento o atributo es definido primero y luego puede ser duplicado en otro lugar de la estructura del documento por el mecanismo de referencia. Los dos elementos o atributos se vuelven de esta forma dos instancias de la misma clase.

Hemos establecido ya que podemos definir elementos y atributos conforme los vamos necesitando (técnica *matriushka*), o crearlos primero y referirlos después (tabla plana). El documento de esquema XML desarrollado por el consorcio W3C nos proporciona un tercer mecanismo usado para definir *dataTypes* (tipos de datos) ya sea simples o de tipo complejo, y utilizar esos tipos para definir atributos y elementos.

Esto se logra dándoles un nombre a los elementos *simpleType* y *complexType* y ubicándolos fuera de la definición de elementos o atributos. Más aún, podemos derivar un tipo de datos *dataType* hacia otro definiendo una restricción sobre los valores de este tipo de datos.

Por ejemplo, para definir un *dataType* llamado "tipo-de-nombre" el cual estará formado por una cadena de hasta 32 caracteres, lo estableceríamos así:

```
<xs:simpleType name="tipo-de-nombre">
    <xs:restriction base="xs:string">
        <xs:maxLength value="32"/>
    </xs:restriction>
</xs:simpleType>
```

El elemento *simpleType* guarda el nombre del nuevo *dataType*. El elemento *restriction* expresa el hecho de que el *dataType* se deriva del *dataType "string"* del espacio de nombres de W3C (*namespace*), pero se le aplica una restricción a sus posibles valores. El atributo *maxLength*, también denominado *faceta* en los documentos del consorcio, especifica cuáles esa restricción, con una longitud máxima de 32 caracteres.

Otra "faceta" poderosa es el elemento *pattern* (patrón), el cual define una proposición regular que debe ser satisfecha. Por ejemplo, si no nos interesa definir el elemento isbn con sus guiones, podríamos definir el *dataType* isbn como:

```
<xs:simpleType name="tipo-isbn">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{X}"/>
    </xs:restriction>
</xs:simpleType>
```

Es decir, lo definimos como una cadena de caracteres cuyo patrón posible son los números 0-9 (cero al nueve) y el "número" X (diez), únicamente. Los guiones quedan por ende excluidos. Si nos interesara agregar el guión como posible en el patrón, simplemente agregamos el valor del guión {-} después del valor del número X; esto es: "[0-9] {X} {-}"

Definir y utilizar *dataTypes* con nombre es comparable a definir una clase y utilizarla para crear un objeto. Un *dataType* es un ente abstracto que puede ser utilizado para definir un elemento o un atributo. El *dataType* realiza entonces con un elemento o un atributo la misma función que una clase realiza con un objeto.

```
<?xml version="1.0" encoding="UTF-7,5"?>
<xs:schema xmlns:xs=
    "http:// www.unam.mx/xmlimaginario/XMLSchema">
    <!— definición de tipos simples —>
    <xs:simpleType name="tipo-nombre">
```

```
    <xs:restriction base="xs:string">
        <xs:maxLength value="32"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="tipo-desde">
    <xs:restriction base="xs:date"/>
</xs:simpleType>
<xs:simpleType name="tipo-descr">
    <xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="tipo-isbn">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{X}"/>
    </xs:restriction>
</xs:simpleType>
<!— definición de tipos complejos —>
    <xs:complexType name="tipo-personaje">
        <xs:sequence>
            <xs:element name="nombre" type="tipo-nombre"/>
            <xs:element name="representa-a" type="tipo-nombre"
                minOccurs="0"
                    maxOccurs="unbounded"/>
            <xs:element name="desde" type="tipo-desde"/>
            <xs:element name="cualificacion" type="tipo-descr"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="tipo-libro">
        <xs:sequence>
            <xs:element name="titulo" type="tipo-nombre"/>
            <xs:element name="autor" type="tipo-nombre"/>
            <xs:element name="personaje" type="tipo-personaje"
                minOccurs="0"/>
            <!— la definición del elemento "personaje" se hace usando el tipo
                complejo "tipo-personaje"—>
            </xs:sequence>
            <xs:attribute name="isbn" type="tipo-isbn"
                use="required"/>
        </xs:complexType>
        <!—Se hace referencia a "tipo-libro" para definir el elemento "libro"—>
        <xs:element name="libro"type="tipo-libro"/>
    </xs:schema>
```

Grupos, compositores y grupos derivados

Los esquemas de acuerdo con el W3C también permiten la definición de grupos de elementos y atributos:

```
<!—definición de un grupo de elementos —>
    <xs:group name="elementos-principales-de-libro">
        <xs:sequence>
            <xs:element name="titulo" type="tipo-nombre"/>
            <xs:element name="autor" type="tipo-nombre"/>
        </xs:sequence>
    </xs:group>
<!— definición de un grupo de atributos —>
<xs:attributeGroupname="atributos-de-libro">
    <xs:attribute name="isbn" type="tipo-isbn"
use="required"/>
    <xs:attribute name="available" type="xs:string"/>
</xs:attributeGroup>
```

Estos grupos pueden ser usados en la definición de tipos complejos, como se muestra:

```
<xs:complexType name="tipo-libro">
    <xs:sequence>
        <xs:group ref="elementos-principales-de-libro"/>
        <xs:element name="personaje" type="tipo-personaje"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    <xs:attributeGroupref="atributos-de-libro"/>
</xs:complexType>
```

Estos grupos no son *dataTypes* sino contenedores de un conjunto de elementos o atributos que pueden ser usados para describir tipos complejos.

Compositores

Hasta ahora hemos visto el compositor de secuencia xs:compositor que define grupos ordenados de elementos o partículas, los cuales pueden ser a su vez grupos de otros compositores. El XML del W3C soporta además dos compositores adicionales que pueden combinarse para permitir varias mezclas. Cada uno de esos compositores puede tener un mínimo y un máximo de ocurrencias (minOccurs y maxOccurs) para definir su cardinalidad. *Unbounded* significa sin límite.

El compositor *xs:choice* describe la elección entre varios posibles elementos o grupos de ellos. El siguiente grupo (los compositores pueden aparecer dentro de grupos tipos complejos u otros compositores) acepta bien un simple nombre de elemento o una secuencia de nombre-de-pila, un apellido-paterno o un segundo-apellido opcional:

```
<xs:group name="tipo-nombres">
    <xs:choice>
        <xs:element name="nombre" type="xs:string"/>
        <xs:sequence>
            <xs:element name="nombre-de-pila" type=
                "xs:string"/>
                <xs:element name="apellido-paterno" type=
                "xs:string"/>
            <xs:element name="segundo-apellido" type="xs:string"
                minOccurs="0"/>
        </xs:sequence>
        </xs:choice>
</xs:group>
```

El compositor *xs:all* define un conjunto no-ordenado de elementos. La siguiente definición de tipos complejos permite a sus elementos contenidos aparecer en cualquier orden:

```
<xs:complexType name="tipo-libro">
    <xs:all>
        <xs:element name="titulo" type="xs:string"/>
        <xs:element name="autor" type="xs:string"/>
        <xs:element name="personaje" type="tipo-personaje"
            minOccurs="0" maxOccurs="unbounded"/>
    </xs:all>
    <xs:attribute name="isbn" type="tipo-isbn"
        use="required"/>
</xs:complexType>
```

Para evitar combinaciones que pudieran resultar ambiguas o demasiado complejas para ser resueltas por las herramientas del esquema XML de W3C, deben agregarse un conjunto de restricciones a la declaración *xs:all*.

- Sólo pueden aparecer como un hijo por una vez al principio del paquete
- Sus hijos pueden ser sólo definiciones *xs:element* o referencias y no pueden tener una cardinalidad mayor que uno.

Derivados de tipos simples

Los *dataTypes* simples pueden ser definidos por derivados de otros *dataTypes*, bien predefinidos e identificados por el namespace del esquema del W3C o definidos en otro lado en el esquema.

Hemos visto ya ejemplos de tipos simples derivados por restricción (usando *xs:restriction*). Las variedades diferentes de restricciones que pueden ser aplicadas a un dataType se llaman facetas, como ya hemos establecido, y son particularmente útiles para establecer la longitud de un elemento, los posibles valores que puede tomar, el número mínimo y máximo de ocurrencias, etcétera.

Existen otros dos métodos de derivación que permiten definir espacios en blanco, listas separadas para ser consultadas, uniones de *dataTypes*, etcétera. La siguiente definición usa *xs:union* para extender nuestro tipo-isbn y que acepte los valores TDB y NA:

```
<xs:simpleType name="tipo-isbn">
    <xs:union>
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:pattern value="[0-9]{X}"/>
            </xs:restriction>
        </xs:simpleType>
        <xs:simpleType>
            <xs:restriction base="xs:NMTOKEN">
                <xs:enumeration value="TBD"/>
                <xs:enumeration value="NA"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:union>
</xs:simpleType>
```

La "unión" ha sido aplicada a dos tipos distintos de valores para permitir la existencia de dos conjuntos de valores; el nuevo *dataType* acepta ahora los valores enumerados TBD y NA.

El siguiente ejemplo "tiposisbn" usa *xs:list* para definir una lista de varios ISBN separados por un espacio en blanco. Deriva también un tipo "tipo-isbn10" usando *xs:restriction* para aceptar entre 1 y 10 valores de números de ISBN separados por un espacio.

```
<xs:simpleType name="tiposisbn">
    <xs:list itemType="tipo-isbn"/9
</xs:simpleType>
<xs:simpleType name="tipo-isbn10">
    <xs:restriction base="tiposisbn">
        <xs:minLength value="1"/>
        <xs:maxLength value="9"/>
    </xs:restriction>
</xs:simpleType>
```

Mezclas en los contenidos

La recomendación del W3C para esquemas nos permite también mezclar partes de documentos marcados con otras secciones sin marcar usando la restricción *xs:complexType*, obsérvese el siguiente texto:

```
<isbn libro="8420639028">
    Obra dramática de José Zorrilla
        Su título, Don Juan tenorio, lo dice todo!
    </isbn libro>
```

Usando la restricción *xs:complexType*:

```
<xs:element name="libro">
    <xs:complexType mixed="true">
        <xs:all>
            <xs:element name="titulo" type="xs:string"/>
            <xs:element name="autor" type="xs:string"/>
        </xs:all>
        <xs:attribute name="isbn" type="xs:string"/>
    </xs:complexType>
</xs:element>
```

Lo cual validará un elemento XML como el siguiente:

```
<isbn libro="8420639028">
    Divertida obra por <autor>José Zorrilla</autor>.
    Su título(<titulo> Don Juan tenorio</titulo>) lo dice todo!
</isbn libro>
```

La construcción *xs:keyref* es súmamante útil en este tipo de obras ya que permite hacer ligas entre algunos elementos y otros. Por ejemplo, deseamos asociar a los distintos elementos "personajes" dentro de esta obra; recordemos la definición establecida en un principio:

```
<personaje>
    <nombre>Don Juan Tenorio</nombre>
    <representa-a>el galan de la obra</representa-a>
    <desde >1844</desde >
    <cualificacion>
        galán extrovertido calavera y pendenciero
    </cualificacion>
</personaje>
```

Y queremos indicar que hay una asociación entre este personaje y otro u otros dentro de la obra; por ejemplo, para que cuando uno sea consultado, se asocie siempre al otro, el cual debe ser referenciado en el mismo libro:

```
<xs:keyref name="referencia-personaje" refer=
    "nombre-personaje">
    <xs:selector xpath="personaje"/>
    <xs:field xpath="representa-a"/>
</xs:keyref>
```

Nos presentará asociaciones con otros personajes de la obra; por ejemplo:

```
<personaje>
    <nombre>Marcos Ciutti</nombre>
    <representa-a > El criado de Don Juan</representa-a>
    <desde>1844</desde>
    <cualificacion>hábil, pícaro, diligente y fiel sirviente de Don Juan
    </cualificacion>
</personaje>
```

Existen todavía algunas otras definiciones para la completa construcción de un esquema, pero no es el caso de este documento consolidarse como un tutorial de esquemas, sino el haberlo ilustrado las considerablemente mayores posibilidades que un esquema permite lograr mucho más allá de la simple definición DTD de un documento. [W3Cschools, 2003]

LENGUAJES PARA ESQUEMAS

Por todo lo anteriormente expuesto sobre la estructura de esquemas, es claro que nos permiten establecer una enorme cantidad de características para el registro, el control y el contexto de un documento dado, pero la estructura puede llegar a complicarse tanto que la construcción de un esquema paso a paso parecería una enorme tarea, como aquí hemos podido observar. Da la impresión de que debe uno memorizar enormes cantidades de instrucciones, símbolos, variables, estructuras, etcétera, lo cual a primera vista parece tedioso y atemorizante. Pero no es así. En la práctica los esquemas no tienden a ser escritos por las personas sino por las máquinas.

Como hemos podido observar, la especificación 1.0 de XML define los conceptos para una correcta formabilidad y validación. A través de ella es muy fácil verificar un documento certificando que tenga buena formabilidad; no obstante, la validación requiere de más trabajo y le permite al usuario definir restricciones más potentes para la estructura del documento. La validación XML requiere que el documento

siga las restricciones establecidas en el DTD, el cual provee el equivalente de lo que sería en un contexto libre una sintaxis para un tipo de documento [Eito, 2001].

En algunos usos los sistemas pueden requerir definiciones de marcado más informativas, o restricciones más estrictas o más flexibles en la estructura del documento, o simplemente diferentes de las expresadas en definiciones DTDs según las define XML 1.0. Hay también una tendencia a permitir marcados y restricciones que puedan ser especificados en una sintaxis totalmente XML, para que de esta forma las diversas herramientas electrónicas para los documentos XML pue dan ser usa das en las mismas especificaciones. Esto puede lograrse gracias a la creación **de lenguajes de programación para la creación e interpretación de esquemas de marcado**.

De la información proveniente de la página del consorcio W3C para los requerimientos de un esquema [Malhotra, 1999] podemos diferir que el propósito del lenguaje de esquemas XML es el de proveer un inventario de las construcciones de marcado XML con las cuales se pueden escribir esquemas. A su vez, el propósito de un esquema es definir una *clase* de documentos XML y usar esas construcciones para delimitar y documentar el significado, uso y relaciones de sus partes constituyentes: tipos de datos, elementos y sus contenidos, atributos y sus valores, entidades y sus contenidos y notaciones, valores implícitos como por ejemplo los de omisión (*default*), etcétera. Los esquemas documentan su propio significado, uso y función. Por ello, el lenguaje de esquemas XML puede ser utilizado para definir, describir y catalogar vocabularios XML para diversas clases de documentos XML. Todo programa XML puede usar el formalismo inherente a un esquema para expresar delimitaciones de valor, estructurales o sintácticas, aplicables a ciertos documentos.

Algunos de los posibles ambientes, además del de las bibliotecas digitales, que podrían verse beneficiados con el uso de la programación basada en estos esquemas pueden ser: el editorial, donde la distribución de información puede involucrar documentos que guardan complejas relaciones entre sí (en estas relaciones los esquemas estructurales describirían las relaciones entre encabezados, nuevos artículos, referencias cruzadas, fotografías, etcétera); el comercio electrónico; la adquisición masiva de datos; la transferencia de información entre bases de datos; el intercambio de metadatos, etcétera.

De acuerdo con el grupo de trabajo de esquemas del consorcio W3C, los principios de diseño para un lenguaje de esquemas se basan en las siguientes características:

El lenguaje para esquemas XML debe:

- Ser más expresivo y detallado que los DTDs.
- Poder expresarse totalmente en XML.
- Ser autodescriptivo.
- Ser utilizable por una amplia variedad de programas que usen XML.
- Poder ser ampliamente utilizable en el Internet.
- Estar optimizado para interoperabilidad.

- Ser suficientemente simple para poder implementarse con recursos moderados.
- Estar coordinado con otras especificaciones W3C (ligas, apuntadores, etcétera).

En cuanto a sus requerimientos estructurales, el lenguaje para esquemas XML debe poder definir los mecanismos para:

- Delimitar la estructura de los documentos (espacios, elementos, atributos) así como de los contenidos (tipos de datos, entidades, notaciones).
- Habilitar la herencia para definir elementos, atributos y tipos de datos.
- Habilitar documentación autocontenida dentro de otra.
- Habilitar descripciones y delimitadores específicos para un programa en especial.
- Contender con la integración de esquemas estructurales con tipos de datos primitivos.
- Definir las relaciones entre el esquema y los documentos XML.
- Definir las relaciones entre esquemas y DTDs.

De lo anterior podemos concluir que en los próximos años habrá una tendencia marcada para construir y desarrollar lenguajes de programación que permitan la creación y explotación de esquemas en una forma mucho más sencilla de la que hemos ido estableciendo aquí en este documento. No parece factible que las personas se dediquen a escribir líneas y líneas con todas esas especificaciones y restricciones. Lo que sí es necesario es que se entiendan los conceptos básicos que permiten definir, restringir, correlacionar, etcétera, a las diferentes entidades de un documento; sería imposible hacerlo de otra forma. Alguien tiene que establecer las características y estructuras de un documento.

Sin embargo la generación en sí del código del esquema será facilitada a través de lenguajes para esquemas. Algunas pantallas amigables al usuario nos permitirán transmitir de manera sencilla la idea que nos hemos hecho del esquema en múltiples instrucciones que serán leídas por otros programas para interpretar y procesar.

Lo importante de todo esto, en suma, es aprender a crear la estructura mental de orden y contexto que un documento conlleva, de acuerdo con la filosofía de los esquemas que, como hemos podido observar, permite contextualizar los documentos mucho más allá de lo que hemos estado acostumbrados. Desde mi punto de vista, el aprender a desarrollar esquemas no estriba en la habilidad de escribir códigos, sino en la habilidad de plantear un documento en la mente bajo una gran estructura organizativa, y creo que eso es lo más relevante del aprendizaje de los esquemas, sobre todo para los bibliotecarios que nos hemos acostumbrado a disectar, catalogar y acomodar los múltiples elementos de una ficha. Me parece que el esquema nos permite aplicar las estructuras de orden y sistema no tan solo a los elementos catalográficos, sino al contenido mismo del documento. De ahí su gran valor como educadores de la mente.

CONCLUSIONES

Como ha podido observarse, el marcado XML, aun en un simple DTD, permite establecer definiciones mucho más completas de un documento que las que podíamos establecer en una ficha catalográfica o un documento HTML para despliegue. Es claro el potencial que el marcado XML nos permite desarrollar en el marcado y la recuperación documental, aun a nivel de un DTD.

Pero, como ha podido comprobarse, las estructuras planteadas por XML pueden irse sofisticando más y más. La propuesta de creación de esquemas relacionados con un tipo dado de documento nos ofrece un potencial aún mucho mayor de registro y explotación de la información, sin perder la calidad de registro catalográfico, como estamos acostumbrados, sino extendiéndolo a niveles más profundos del contenido.

Los esquemas pueden viajar en la red junto con los documentos marcados bajo su estructura. Esto quiere decir que el nivel y calidad de nuestro marcado no está constreñido a nuestro sistema de cómputo. Podemos entregar e intercambiar documentos con estructuras muy complejas y precisas que vayan llevando a niveles más profundos, profesionales y satisfactorios de almacenamiento y recuperación de información.

Como puede deducirse es importante ir creando proyectos que desarrollen esquemas específicos para distintos tipos de documentos dentro de las bibliotecas mexicanas. Ello tendría un gran valor a futuro para las nacientes bibliotecas digitales de nuestro país, ya que podría ofrecerse un estándar mexicano y, sin embargo, perfectamente internacional, hecho por profesionales, y con el cual puedan esas bibliotecas empezar a "marcar" sus documentos de acuerdo con la sugerencia de cada esquema. En primer lugar, podría resolverse de una vez la discusión teórica sobre si usar DTDs o esquemas; o DTDs y esquemas. Con ese resultado, y dependiendo del tipo de material, cada biblioteca podrá entonces alojar documentos en bibliotecas digitales de tipo "novela", "obra de teatro", "libro de texto", etcétera. A la larga, el proyecto podría continuarse para obtener un conjunto de todos los formatos "típicos" de documentos: libros, revistas, mapas, discos, películas, etcétera, como alguna vez MARC lo hizo, pero con divisiones específicas para cada variedad de documento en cada formato, y con alcances para cada parte del contenido y no sólo para los datos de referencia. Esto aseguraría un buen nivel de marcado, la estandarización entre diversas bibliotecas digitales mexicanas que se vayan creando y, lo más importante, una espléndida y poderosa recuperación y explotación por parte de los usuarios, etcétera. Con ello podrían sentarse las bases para un modelo de desarrollo de esquemas que pudiese ser llevado a nivel de recomendaciones para el medio mexicano y tal vez ser imitado en otros países latinoamericanos.

Si no iniciamos un proyecto de este corte, en un futuro cercano las bibliotecas empezarán a hacer sus marcados XML con forme al diseño de cada una de ellas, y como no existe ningún estándar, habrá muy diferentes especificaciones y calidades de marcado, incluso algunas que nada tengan que ver con este sector. Esto no sería bueno para esas colecciones digitales, pues cuando se deseara adoptar algún estándar preestablecido

quizá sería tarde y podrían existir muchos documentos ya introducidos en alguna colección dada, lo cual dificultaría la conversión a nuevos estándares y obstaculizaría la estandarización de marcados y calidades homogéneas para las colecciones electrónicas. Peor aún sería tener que esperar e imitar estándares internacionales, que si bien pueden estar bien hechos nos siguen dejando a la sombra de esos desarrollos hechos en otros países, no del todo acordes con nuestra realidad nacional.

Finalmente, las conclusiones pueden resumirse de acuerdo con lo siguiente:

- Los lenguajes de marcado realmente no son nuevos; tienen más de cuatro décadas y lo que observamos hoy son sus versiones más sofisticadas.
- Los metalenguajes usados hoy en día en el mundo electrónico-documental tienen diversas características, ventajas y desventajas dependiendo del contexto de su creación y su momento histórico.
- La tendencia hoy en día es que los lenguajes de marcado ayuden a consignar no tan solo los aspectos catalográficos de una obra, sino también los relacionados con su contenido.
- XML se constituye como la versión más completa y funcional de los lenguajes de marcado hoy en día.
- Los esquemas permiten la definición de estructuras y relaciones mucho más completas que las de un DTD.
- El establecimiento de lenguajes de creación de esquemas facilitará la parte operativa y mecánica del desarrollo de esquemas.
- Es importante ir creando definiciones, DTD o esquemas, de las principales clases de documentos en el medio bibliotecario mexicano, para garantizar su calidad y profundidad.
- Es importante entender la filosofía de creación de esquemas como un mecanismo de ejercicio mental de organización y sistematización de la información.

REFERENCIAS BIBLIOGRÁFICAS[1]

A general Introduction to the MARC Format. Disponible en: http://www.tlcdelivers.com/tlc/crs/Gen0009.htm

Campbell, Ch. 2003. *"XML Schema"*. En: Sigmond Record, Vol. 32, No 2, Junio 2003. pp. 96-101.

Eito, R. 2001. Programación con XML. Madrid: Anaya. ISBN: 84-415-1186-1. pp. 487-540.

* Extensible Markup Language (XML) 1.0 (2nd. ed.), W3C Recommendations. Octubre 6, 2002. Disponible en: http://www.w3c.org/TR/1998/NOTE-XML-data-0105/

Gutierrez, A y Martínez, R. 2001. XML a través de ejemplos. Madrid: RAMA. ISBN: 84-7897-455-5.

1 Los URL'S marcados con un * corresponden a un sitio oficial o con nivel de estándar internacional.

Free Dictionary of computing. 2000. Entrada por: "Backus-Naur Form". Disponible en: http://burks.bton.ac.uk/burks/foldoc/

* Iflanet. *What is MARC*. Disponible en: http://www.ifla.org/VI/3/p1996-1/unimarc.htm

* Library of Congress. MARC Standards. 2003. Disponible en: http://www.loc.gov/marc

* Library of Congress. MARC 21 XML Schema. 2003. Disponible en: http://www.loc.gov/standards/marcxml

Morrison, M. 2002. Teach Yourself XML in 24 hours. Indiana: Sams. ISBN: 0-672-32213-7. Parte II, hora 5, "Using XML Schema", pp. 69-95.

MSDN online. 2000. XML Developing Center Disponible en: http://msdn.microsoft.com/library/default.asp?url=/nhp/Default.asp?contentid=28000438

* SGML ISO 8879. 1986. Disponible en: http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=16387

Should it be an element or a type ? Disponible en: http://www.xfront.com/ElementVersusType.html

Teasdale, G. 1995. L'Hypertexte: historique et applications en bibliothéconomie. Université de Montréal. Disponible en: http://www.infotheque.info/ressource/942.html

Timeline of XML Events. Disponible en: http://www.ukoln.ac.uk/web-focus/events/workshops/webmaster-2002/materials/savory/slides/img18.html

* XHTML 1.0: The eXtensible HyperText Markup Language. Disponible en: http://www.w3.org/TR/xhtml1/#xhtml

The XML Cover Pages - Home page. Disponible en: http://oasis-open.org/cover/

[Malhotra, A. 1999]. *XML Schema Requirements*. Disponible en: http://www.w3.org/TR/NOTE-xml-schema-req

Van der Vlist, E. 2001. *Using W3C XML Schema*. Disponible en: http://www.xml.com/pub/a/2000/11/29/schemas/part1.html

Thompson, H. 1997. *Why I demand "Schemata"*. Disponible en: http://www.ltg.ed.ac.uk/~ht/sgml97.html

* W3C Schools. 2003. XML Tutorial. Disponible en: http://www.w3schools.com/xml/default.asp

Young, M.J. 2002. XML Step by Step. 2ª ed. Redmond: Microsoft Press. ISBN: 0-7356-1465-2. Capítulo 7: "Creating valid XML documents using XML Schemas". pp. 163-192.