Especificaciones XML aplicadas a la Documentación

José A. Senso
Facultad de Biblioteconomía y Documentación
Universidad de Granada
jsenso@platon.ugr.es

Antonio de la Rosa
Groningen Graduate School for Behavioral and Cognitive Neurosciences.
a.de.la.rosa.pinero@let.rug.nl

RESUM
L'evolució lògica de la tecnologia i el mercat fa que la integració del món documental amb el Web (o amb qualsevol altre sistema d'informació que es basi en la mateixa filosofia (p.ex. una intranet) sigui un fet gairebé imminent. La nostra tasca com a professionals de la informació és pensar la millor forma d'implementar els processos de treball documentals a aquest entorn.
L'aparició de XML (eXtensible Markup Language), versió abreujada de SGML (Standard Generalized Markup Language), i de les seves especificacions fa pensar en que estem davant d'un conjunt d'eines adequat per assolir aquest objectiu.
Aquest treball analitza les característiques principals d'aquest llenguatge, així com la d'altres generats paralel·lament, i ofereix algunes possibles solucions als problemes plantejats en el camp de la documentació electrònica.

RESUMEN
La evolución lógica de la tecnología y el mercado hace que la integración del mundo documental con el Web (o con cualquier otro sistema de información que se base en la misma filosofía (p. ej., una intranet) sea un hecho casi inminente. Nuestra tarea como profesionales de la información es pensar la mejor forma de implementar los procesos de trabajo documentales a este entorno.
La aparición de XML (eXtensible Markup Language), versión abreviada de SGML (Standard Generalized Markup Language), y de sus especificaciones hace pensar que estamos ante un conjunto de herramientas apropiado para lograr este objetivo.
Este trabajo analiza las características principales de este lenguaje, así como las de otros generados paralelamente, y ofrece algunas posibles soluciones a los problemas planteados en el campo de la documentación electrónica.

ABSTRACT
The logic evolution of the technology and the market makes the integration of the documental world into the web (or into wichever information system based on the same phylosophy, e.g. an intranet) a fact almost imminent. Our duty as information professionals is to think the best way to implement the procedures of the documental task into this environment.
The appearence of the XML (eXtended Markup Language), short version of the SGML (Standard Generalized Markup Language), and its specifications makes consider we are in front of the right tools to get this aim.
The present survey analyses the main characteristic of this language, as well as the ones of other languages created parallelly, and offers some possible solutions to the problems stated in the electronic documentation field.

1. Estructuración

La clave para una gestión óptima de la documentación está en la estructuración de los datos. Hasta ahora, los únicos sistemas con los que contábamos para lograr esto eran los que nos proporcionaban las bases de datos (salvo, claro está, los que utilizaban SGML). La mayoría de estas soluciones proponían el uso de formatos propietarios para la gestión completa de la información (creación, almacenamiento…).

Este sistema facilita el trabajo diario y, en cierta forma, permite la automatización de determinados procesos. Pero algo que resulta chocante es que la Documentación, que siempre se ha caracterizado por su interés hacia la homogeneización (de otra forma no se entenderían los esfuerzos normalizadores o, sin ir más lejos, la existencia del formato MARC), ahora se vuelque en el uso de formatos que dependen de sistemas operativos, versiones o plataformas concretas. A todas luces, esto parece ilógico.

Es muy posible que el motivo por el cual todavía no se ha dado el salto hacia la multiplataforma sea el limitado número de soluciones que ofrecen los sistemas existentes. HTML (Hypertext Markup Language) sólo permite definir el formato del texto, nunca su contenido, y SGML es demasiado complejo como para que su uso resulte rentable.

Posiblemente XML facilite este cambio. Si XML no tiene la solución a un problema planteado por la Documentación es muy probable que alguna de sus especificaciones sí. Veamos algunos ejemplos:

- Uno de los principales inconvenientes generados por la proliferación de colecciones de documentos electrónicos, así como la gestión de bibliotecas digitales, está en la localización y recuperación de este tipo de información. El uso de sistemas de metadatos que se encarguen de su descripción puede facilitar, en gran medida, este trabajo. Resource Description Framework (RDF) permite la definición de conjuntos de metadatos dentro del entorno de trabajo de XML. Otros sistemas como Meta Content Framework (MCF) o Encoded Archival Description (EAD) pueden ser también mecanismos válidos para lograr estos objetivos.

- Las aplicaciones para el comercio electrónico inciden directamente en la profesión, facilitando el intercambio electrónico de documentación y la gestión de pedidos de originales. Para llevar a cabo estos procesos XML propone dos vías de actuación: por un lado especificaciones que proporcionan formatos normalizados de trabajo –Open Financial Exchange (OFE) u Open Trading Protocol (OTP)– y, por otro, la integración de protocolos ampliamente utilizados por la comunidad internacional (como es el caso de EDI –Electronic Data Interchange–) dentro del mismo lenguaje XML, dando lugar al XML/EDI.

- En esta línea, los mecanismos utilizados para el intercambio de información de cualquier tipo (especialmente académica) se verán reforzados con el uso de IMS Metadata Specification y Tutorial Markup Language (TML).

- La utilización de información multimedia siempre ha planteado una gran cantidad de dificultades en los centros de documentación. Al problema siempre presente de su almacenamiento se le une el de la gestión, la localización, la recuperación o la visualización en tiempo real a través de un entorno distribuido. Synchronized Multimedia Integration Language (SMIL) o Precision Graphics Markup Language (PGML) son dos especificaciones XML que facilitan en gran medida la realización de estos procesos.

- En cuanto un centro de documentación se plantea trabajar con información científica en formato electrónico aparece un problema constante: ¿cómo introducir los datos matemáticos o las fórmulas? XML proporciona una solución diferente para cada área del conocimiento: Mathematical Markup Language (MathML), Chemical Markup Language (CML), Telecommunication Interchange Markup (TIM), etc.

- Dentro del campo de la recuperación de la información XML cuenta con eXtensible Structured Query Language (XSQL), lenguaje que utiliza sentencias SQL para consultar datos XML o Vector Markup Language (VML) como base para aplicar el modelo de espacio vectorial.

Parece evidente que XML cuenta con suficientes recursos como para solventar cualquier dificultad que se pueda plantear dentro del campo de la Documentación. ¿Cuál es la clave para su aplicación?

Sin duda alguna la respuesta está en la estructuración de la información. Las herramientas para crear documentos XML (y sus especificaciones) se basan en estándares abiertos. Es decir: el mercado no tiene ningún tipo de control sobre ellos y, por lo tanto, no puede imponer dependencia alguna hacia una determinada marca o sistema operativo. Por otro lado es más sencillo el intercambio de información entre organizaciones.

Dentro del proceso del diseño de un sistema de información que utilice XML es necesario cuantificar el nivel de estructuración de los datos que se necesita y el porqué. Esto debe llevarse a cabo teniendo en cuenta las posibilidades de navegación, la cantidad de procesos que se pretenda automatizar, el control de calidad (con vistas a lograr la tan famosa y deseada certificación ISO 9000), la actualización de la información, etc.

Una de las disciplinas donde es más fácil la implementación de XML es la Documentación. XML ordena la información en estructuras arbóreas, creando diversas jerarquías que facilitan su gestión. Los productos documentales en general presentan una estructura implícita que facilita ese este tipo de planteamiento.

De hecho, dentro del mundo de la Documentación existe un sistema de intercambio de datos conceptualmente próximo a él: el formato MARC. Hagamos una comparación entre ambos por medio de un sencillo ejemplo. El siguiente registro MARC:

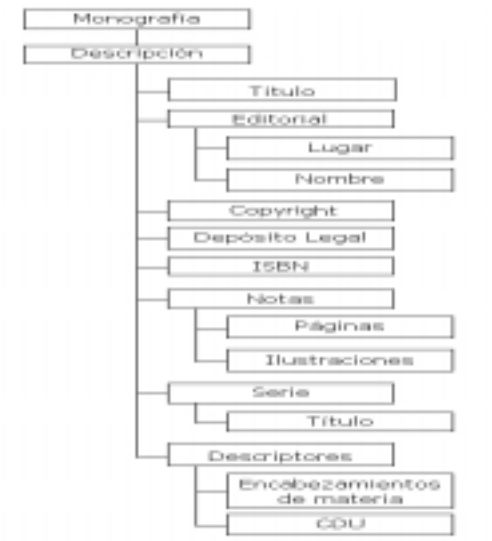
130	30	$a	El 4 – 4
245	13	$a	El 4 – 4 $b: la aventura del todoterreno
250		$a	1ª ed.
260	0	$a	Barcelona $b Plaza & Janés $c 1996
300		$a	105 p. $b il. $c 21 cm
440	0	$a	Deportes de aventura
650	08	$a	Automovilismo
650	08	$a	Vehículos todo terreno
740	31	$a	El cuatro por cuatro

tendría esta correspondencia en XML:

```
<?XML version=»1.0»?>
<MONOGRAFIA>
    </libro fechalibro=»1975»>
            <descripcion>
                    <!—1ª edición en 1996 —>
                    <titulo> El 4 _ 4: la aventura del todoterreno</titulo>
                    <editorial>
                            <lugar>Barcelona</lugar>
                            <nombre>Plaza & Janés</nombre>
                    </editorial>
                    <copyright>1996</copyright>
                    <deplegal>B 18092-1996</deplegal>
                    <isbn>84-01-57002-6</isbn>
                    <notas>
                            <paginas>105</paginas>
                            <ilustraciones>Sí</ilustraciones>
                    </notas>
                    </serie>
                            <titulo>Deportes de aventura</titulo>
                    </serie>
                    <descriptores>
                            <ul-encab>
                            <li>Automovilismo</li>
                            <li>Vehículos todo terreno</li>
                            </ul-encab>
                            </ul-cdu>
                            <li>796.71</li>
                            <li>629.373</li>
                            </ul-cdu>
                    </descriptores>
            </descripcion>
    </libro fechalibro=»1996»>
</MONOGRAFIA>
```

Se observa cómo MARC pretende estructurar la información por medio de códigos, pero el resultado carece de flexibilidad. Por otro lado no permite que el registro pueda ser considerado como un objeto, sino como un texto plano. A todo esto se añade la mínima posibilidad de integración con el Web.

El registro «catalogado» por medio de XML es más flexible (permite representar el contenido completo del documento simplemente añadiendo un nuevo conjunto de etiquetas –resumen, capítulo 1, párrafo,etc.–) y, además, facilita su ordenación generando un árbol:



En líneas generales, se podría hablar de tres tendencias que afectan en la actualidad a la documentación electrónica (sobre todo a la documentación presente en el Web) sobre las cuales se está desarrollado toda una nueva generación de normas y tecnologías. Dichas tendencias son: *presentación* de la información (CSS, XSL, etc.), *gestión* (RDF) y *distribución* (P3P). Todas estas herramientas están enmarcadas dentro de un entorno cuya filosofía de trabajo se centra en la estructuración e intercambio de datos como si fueran objetos. Para lograr este fin se trabaja desde tres frentes distintos: XML, HTTP-NG y DOM.

2. XML

Como se puede deducir de lo expuesto hasta ahora, XML es un lenguaje de etiquetado que sirve para la representación digital de documentos (y puede incluir tanto texto como fotografías, gráficos…). Se creó para hacer posible el intercambio de documentos estructurados a través del Web.

Un documento XML tiene un contenido (conjunto de caracteres Unicode) delimitado por una serie de etiquetas. Estas etiquetas pueden representar las siguientes categorías:

- Elemento: es la pieza clave en los lenguajes de etiquetado. Están delimitados por < y > para el comienzo del elemento y por </ y > para el final. Definen la naturaleza del contenido que etiquetan.

- Atributo: especifica las cualidades del elemento (en el elemento TEMARIO, NÚMERO=»2» sería el atributo). Siempre aparecen entre comillas.

- Referencia de entidad: es la forma alternativa que tienen los lenguajes de etiquetado de representar algún carácter reservado para usos concretos (por ejemplo, las comillas dobles para representar atributos).

- Comentarios: es un «residuo» de los lenguajes de programación. Sirven para explicar algún apartado concreto del documento. Comienzan y terminan siempre de la misma forma (<!—y —>), de tal forma que el navegador los puede ignorar cuando interprete esa etiqueta.

- Instrucción de proceso: sirven para suministrar información adicional a las aplicaciones. Funcionan de la misma forma que los comentarios, con la salvedad de que sí son tenidos en cuenta por los navegadores.

- CDATA: se utiliza para especificar al parser (analizador sintáctico) que ignore un determinado conjunto de etiquetas.

Algunas de las características prácticas de XML que deben tenerse en cuenta son:

- En principio, no necesita DTD (Document Type Definition). La DTD está formada por una serie de definiciones con los tipos de elementos, atributos y entidades que se consideran «válidas» dentro del documento XML. También especifica el orden en que pueden aparecer. Si no se usa DTD, XML mantiene punteros a la estructura de datos, lo que simplifica el desarrollo de la creación de la aplicación y ahorra tiempo de ejecución.

- Tipo de letra: XML distingue entre mayúsculas y minúsculas a la hora de trabajar con elementos. Si se ha especificado en la DTD el elemento <TEMARIO> y en el documento se escribe <Temario> el parser avisará que se está produciendo un llamamiento erróneo.

- Requiere documentos «bien formados». En un documento XML siempre se necesitará una etiqueta de principio y otra de final para elemento. Recordamos que, por ejemplo, en HTML esto no es necesario, y que muchos documentos que no delimitan el final de un elemento pueden ser interpretados sin ningún problema por cualquier navegador.

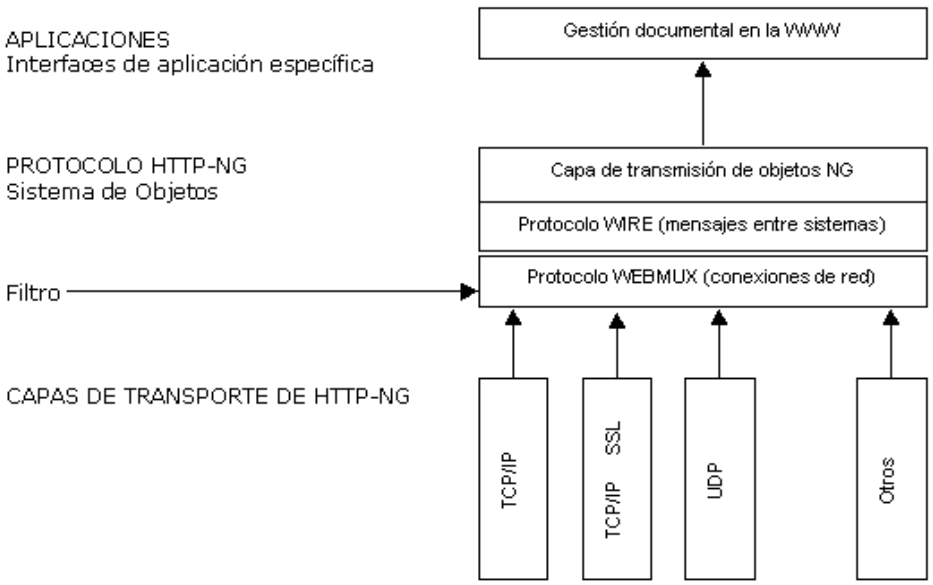
- No permite definir como entidad interna información especial (fórmulas matemáticas, símbolos químicos…). Para paliar esta deficiencia han surgido especificaciones como MathML o TIM.

- Una etiqueta determinada siempre realizará la misma función.

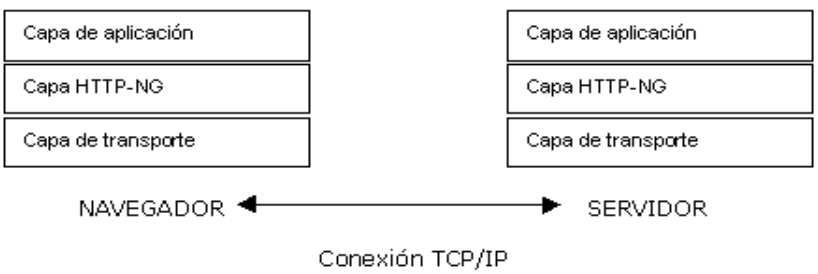
- No se pueden enumerar los componentes de un elemento en cualquier orden, existe un orden preestablecido.

2.1. HTTP-NG

El propósito de HTTP-NG (Hypertext Transfer Protocol – Next Generation) es diseñar, implementar y probar una nueva arquitectura, basada en un modelo de orientación a objetos, que pretende ser simple, extensible y distribuida. HTTP-NG incluye un protocolo encargado de gestionar las conexiones de red (WEBMUX), otro para la transmisión de mensajes entre sistemas (WIRE) y un conjunto de métodos, interfaces y objetos diseñados para adaptarse a la navegación a través del WWW. El siguiente esquema representa las capas que debería tener cualquier sistema compatible con HTTP-NG.



Este esquema se tendría que dar en los dos participantes típicos de una transacción Web: el navegador y el servidor:



A primera vista, tanto *navegador* como *servidor* parecen implementar HTTP-NG de la misma forma, pero si se examinan con un poco más de profundidad pueden verse las diferencias que corresponden al papel de cliente del *navegador* y servidor (valga la redundancia) del *servidor*. Por ejemplo, los objetos previstos por HTTP-NG son implementados mediante clases sucedáneas en el *navegador* y mediante clases auténticas en el *servidor*. Del mismo modo, el *servidor* presenta mayor variedad de protocolos y elementos de transporte puesto que tiene que negociar con una mayor variedad de posibles clientes.

En general, el desarrollo de la gestión de información (tanto la gestión del conocimiento como otros aspectos más técnicos: desarrollo de interfaces o la automatización de procesos) presenta más posibilidades de trabajo si se orienta hacia la búsqueda de modelos flexibles que permitan la estructuración e intercambio de información electrónica dentro de un entorno orientado a objetos.

2.2. DOM (Document Object Model)

No se trata de una especificación con la que poder trabajar directamente con datos, como es el caso de XML. Es más bien un modelo de referencia, de base, para generar estructuras y contenidos que puedan ser utilizados en HTML (a partir de su versión 4.0) y XML (a partir de su versión 1.0).

Mediante este modelo es posible crear objetos en documentos XML y HML y, posteriormente, relacionarlos entre sí.

DOM representa un documento como una colección jerárquica de nodos (objeto, atributo, texto, comentario e instrucciones de proceso). Además, define una serie de entidades que tienen como objetivo el enriquecer la información de cada nodo.

Estos nodos se integran entre sí por medio de relaciones padre-hijo. El nodo raíz es el único que no tiene un elemento superordenado o padre, mientras que los demás presentan generalmente un elemento «padre» y uno o varios subordenados «hijos» (también se dan elementos sin hijos).

3. Especificaciones XML

3.1. XSL (eXtensible Style Language)

El carácter de XSL es dual. Por una parte se trata de un lenguaje que sirve para transformar los documentos XML, por otra es un conjunto de etiquetas, elementos y objetos XML que especifican una semántica de formato o presentación de documentos. De cara a la Documentación, ambas cuestiones son fundamentales.

Entre las principales ventajas de XSL destacan:

- Mantener un formato de salida de un documento independientemente de su estructura y de cualquier otra especificación.
- Reutilizar el mismo estereotipo de formato para crear diversos documentos.
- Favorecer la automatización de tareas, ya que es compatible con lenguajes de programación. Especialmente con ECMAScript.

XSL es una especificación desarrollada por el W3C con el fin de aplicar formatos (mediante hojas de estilo) a los documentos XML de manera estandarizada. XSL está fuertemente ligado a tres especificaciones:

– DSSSL (Document Style Specification Language ISO/IEC 10179). Norma para crear hojas de estilo que se utiliza para dar formato a documentos SGML. Se trata de un lenguaje muy potente, extenso y complejo. En los primeros tiempos de XML se acordó utilizarlo (convenientemente simplificado como DSSSL-O) como un lenguaje de estilo para XML y, de hecho, muchas características de XSL son herencia de DSSSL.

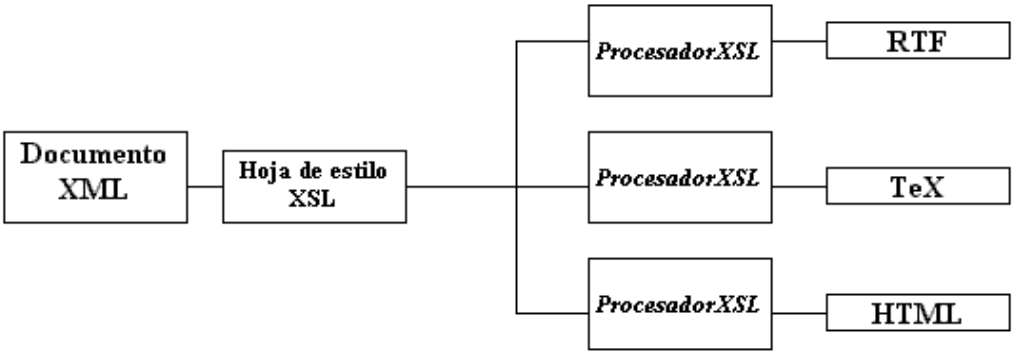
– CSS (Cascading Style Sheet). Uno de los objetivos de desarrollo de XSL es que las hojas de estilo sean independientes del formato. CSS fue diseñado para intentar separar, en la medida de lo posible, contenido y formato en documentos HTML. Sin embargo presenta algunas limitaciones que XSL supera.

– HTML. Hoy por hoy es el lenguaje con mayor presencia en el Web. XSL reconoce esta circunstancia y la importancia de generar HTML a partir de XML. Por este motivo la especificación incorpora varias extensiones que facilitan la generación de código HTML.

A pesar de que XSL se basa en DSSSL, y de que sus conceptos básicos sean similares a los de la norma internacional, los objetivos fundamentales sobre los que se centró el desarrollo de esta especificación eran los de simplicidad[1] y su orientación (Internet).

El borrador que se envió en 1997 al W3C estaba avalado por expertos de Microsoft, INSO y Arbortext entre otros. Quizá por ese motivo, ésta ha sido una de las especificaciones XML sobre la que se han diseñado más aplicaciones: el parser MSXSL de Microsoft o ADEPT-7 y XMLStyler de Arbortext.

XSL ha sido diseñado para ser independiente del formato de salida, es decir, una hoja de estilo XSL puede usarse para obtener un texto con formato RTF (Rich Text Format), TeX o HTML. Esto significa que es muy sencillo reutilizar un mismo documento tantas veces como se desee. El siguiente gráfico representa este esquema de trabajo:



Una hoja de estilo XSL se compone de un conjunto de reglas, llamadas reglas de construcción, que especifican cómo se debe convertir el código que un documento XML en una jerarquía de objetos de formato. El procesador XSL distribuye el contenido XML del documento fuente dentro de esos objetos.

Al igual que sucede con los elementos XML, los objetos de formato pueden anidarse unos dentro de otros (y de esa forma constituir una jerarquía). Por ejemplo, un objeto página puede contener un objeto tabla que, a su vez, puede contener un objeto celda, etc. XSL define una serie de objetos que deben ser el núcleo de todas las implementaciones XSL.

1. De hecho, XSL se define a sí mismo como un lenguaje «declarativo». En programación se denominan así los lenguajes que utilizan declaraciones para expresar mandatos. Estos lenguajes son más fáciles de aprender ya que no requieren operaciones complejas.

Estos objetos tienen asociadas una serie de características, y son esas características las que sirven para controlarlo. Así, el ancho sería una característica del objeto página, y la justificación lo es del objeto párrafo, etc.

Además de objetos de formato y de reglas de construcción, XSL proporciona una serie de reglas de estilo que se encargan de aplicar ciertas características de formato comunes a los objetos. Estas reglas de estilo heredan de CSS la forma de aplicación en «cascada», es decir, que se pueden aplicar múltiples reglas de estilo al mismo elemento.

En general, la forma que adopta una regla de construcción es:

```
<regla>
    [patrón]
    [acción]
</regla>
```

El patrón se utiliza para especificar sobre qué elemento o elementos se va a aplicar la regla. La acción indica qué va a hacer la regla.

Veamos las partes fundamentales en un ejemplo donde crearemos una hoja de estilo que determine alguna de las características del título y los párrafos y los títulos:

```
<xsl>
    <regla>
        <target-element type=»..p» />
            <paragraph font-size=»10pt»>
            <paragraph font-face=»Arial»>
            </paragraph>
    </regla>

    <regla>
        <target-element type=»title» />
            <sequence font-size=»18pt» font-face=»Verdana» font-weight=»italic»>
            </sequence>
    </regla>
</xsl>
```

Se debe mencionar el gran parecido que tiene esta expresión con la que se podría dar en una CSS para HTML:

```
        <style>
P           {
        font-size: large;
        font-family: Arial;
        font-weight: normal;
}
H1          {
        font-size: x-small;
        font-family: Verdana;
        font-weight: bold;
}
        </style>
```

En el ejemplo de XSL, cada una de las reglas contiene una acción (poner en negrita, utilizar un tipo de letra y un tamaño diferente…) y un target-element, es decir, el elemento al que se le aplica dicha acción. Tal y como está estructurada la orden, siempre que el visualizador encuentre el elemento p en un documento XML deberá generar un párrafo con una letra con determinadas características.

XSL suministra un conjunto de objetos de flujo[2] que sirven para representar los diversos componentes del documento reproducido. En una hoja de estilo encontramos objetos de flujo simples (párrafos, tablas…) y complejos (gráficos, líneas horizontales…).

Lo realmente importante de los objetos de flujo (además de lo obvio: la representación del texto) es que sirven para formar árboles, es decir: el documento en sí es el tronco; párrafos, tablas, viñetas, listas numeradas, etc. forman las ramas; y los caracteres, imágenes y demás información atómica[3] serían las hojas.

Este árbol de objetos de flujo nos puede servir para establecer una relación con el árbol de análisis de un documento. Lo que puede ser importante, en un primer momento, para, por ejemplo, generar el índice o tabla de contenidos. Pero se puede ir más allá.

Es lógico pensar que cada objeto de flujo posee características propias, que dependerán de la clase a la que pertenezcan (todos los párrafos, todos los títulos…). De esta forma podemos estructurar con mayor precisión la información. De hecho, si unimos un esquema de trabajo (XSL) con un sistema estructurado para representar la información (XML) es posible empezar a pensar en la posibilidad de generar ciertos productos documentales de forma más o menos automatizada (especialmente resúmenes y catalogaciones), especialmente utilizando las posibilidades que ofrecen los patrones.

3.1.1. Patrones

Mediante un patrón es posible seleccionar los elementos a los que se le deben aplicar unas determinadas reglas de estilo. Como hemos visto en el ejemplo, cada patrón tiene un target-element, que es donde se especifican los elementos que deben buscarse en el documento. Al mismo tiempo es posible utilizar el atributo type, que permite discriminar la búsqueda sobre la base de un determinado tipo de etiqueta o aplicar una regla de estilo a elementos de cualquier tipo (regla por defecto), a los que posean determinados atributos, etc. Para hacer esto último XSL utiliza un elemento que cuenta con tres atributos: name, value y has-value.

Lo realmente significativo del trabajo con patrones es la posibilidad de realizar correspondencias en función del contexto de un elemento. Por ejemplo, si se desea especificar que el título de un trabajo debe corresponder al área 1 de la ISBD, utilizaremos la siguiente regla:

```
<modification>
        <titulo>
                <target-element type=»..t» />
        </titulo>
        <sequence extra-value=»área 1»>
</modification>
```

Los patrones XSL son muy potentes, pero cuando demuestran esa potencia en la práctica es cuando los asociamos a determinadas acciones. Esto nos da pie a pensar que, realmente, la posibilidad de generar de forma automática determinados productos no es tan descabellada como puede parecer. Sobre todo si tenemos en cuenta que podemos utilizar XML para llegar donde no se alcanza con XSL.

3.1.2. Acciones y ECMAScript

Con XSL tenemos un mecanismo que nos facilita trabajar con precisión sobre las características de cada uno de los elementos del documento XML. Ahora sólo necesitamos una

2. Elementos como los párrafos, las listas numeradas, los títulos, etc. se denominan objetos de flujo porque el texto fluye de un lado a otro mientras ellos se quedan «quietos» determinando posiciones.

3. Se denominan atómicos porque no están formados por otros objetos.

herramienta que nos permita incorporar algún tipo de comportamiento a esos datos. Para ello podemos utilizar acciones. Una acción XSL permite trabajar con un conjunto de datos (y por tanto de objetos de flujo) y generar un árbol de salida tras haber realizado modificaciones en ese conjunto (edición de datos, cambio de atributos…). De esta forma las acciones generarán objetos de flujo resultado de una operación posterior y especificarán al visualizador XML cómo debe procesar el contenido de ese objeto.

La gestión de esta información no es tan fácil como parece, ya que la mayoría de objetos tienen un contenido concreto (párrafo=texto, título=texto). Este contenido se denomina hijo, y se puede procesar utilizando el elemento children.

Es cierto que XSL no está diseñado para gestionar cadenas de caracteres, pero eso no significa que no pueda hacerlo. No por sí sólo, pero sí ayudado por un lenguaje de programación sí. XSL incorpora el lenguaje ECMAScript[4] que permite buscar, seleccionar y recuperar secciones concretas de un documento con las que luego se puede trabajar a parte (con el fin de modificar sus características, cambiar o eliminar cadenas específicas de caracteres, etc.).

Hemos visto hasta aquí algunas de las posibilidades de XSL. Básicamente, XSL es una DTD XML que ofrece una solución al problema del formato de la información aprovechando características intrínsecas de XML (sobre todo la estructura arbórea y la tendencia a identificar elemento con objeto).

3.2. XLink (XML Linking Language)

Contar con un mecanismo que permita enlazar los documentos entre sí utilizando el paradigma hipermedia es vital en cualquier sistema de información. En la actualidad el elemento más utilizado para realizar esta labor es el Anchor del HTML. Sin embargo, este instrumento es unidireccional (envía de un origen a un destino) y tiene más características de un enlace (muestra una relación entre dos recursos) que una dirección (describe cómo encontrar los dos recursos).

Para realizar la tarea de direccionamiento XML utiliza la especificación XLink, creada a partir de otras normas como, por ejemplo, el mecanismo de enlaces de TEI (Text Encoding Initiative) o el estándar que usa SGML para formalizar este mismo proceso: Hytime (ISO/IEC 10744).

3.2.1. Sistema de enlaces de XLink

Todos los enlaces tienen una serie de características comunes: relación (posibilidad de expresar diferentes tipos de relaciones entre recursos), topología, sintaxis, presentación y comportamiento (forma de actuar cuando se activa). Esta última posibilidad, la del comportamiento, es muy importante de cara a la integración en sistemas orientados a objetos.

XLink permite generar *enlaces simples* que tienen la misma funcionalidad que los ya mencionados elementos anchor de HTML: unir un documento (origen) con otro (destino). Si en HTML estos enlaces utilizan siempre el mismo elemento (<A>), en XLink se pueden crear diferentes tipos de elementos, cada uno de ellos con sus propios atributos, declaraciones y comportamientos. Por ejemplo, es posible construir un conjunto de enlaces simples para los diferentes componentes de un documento electrónico (notas al pie, citas bibliográficas, leyenda de las imágenes…):

4. Variante del JavaScript. A pesar de tener un núcleo común, ECMAScript añade características propias para gestionar mejor el contenido de los documentos XML.

```
…ejemplo de nota al final de una página:
(<nota al pie xml:link=»simple» href=»http:www.ugr.es/doc1/footnote.xml»>1</link>)
…ejemplo de una referencia bibliográfica:
(<cita xml:link=»simple» href=»http://www.ugr.es/doc1/citation.xml»>Salton</link>, 1978)
```

El concepto de *enlace extendido* se aplica a todos aquellos enlaces con más de un destino, es decir, los que pueden implicar a cualquier número de recursos y subrecursos. Un ejemplo:

```
<nota al pie xml:link:»extended»>
        <locator href=»salton.xml role=»Salton» />
        <locator href=»belkin.xml role=»Belkin» />
<Par>Los sistemas de recuperación de información no han sufrido cambios…</Par>
</nota al pie>
```

Este documento especifica más de una dirección de destino, de tal forma que el lector podrá escoger de un menú la cita que más le interese consultar. La cita seleccionada se presentará de diversas formas dependiendo del tipo de comportamiento especificado (nueva ventana, reemplazando un enlace por otro, generando un nuevo enlace…).

En este punto hemos incluido dos elementos que no aparecen en los enlaces simples, sencillamente porque entonces dejarían de serlo: los localizadores y los roles. Cada localizador «apunta» a un recurso concreto, mientras que los roles especifican la semántica a utilizar durante el proceso. De hecho, varios localizadores pueden tener el mismo rol.

XLink permite, además, agrupar los enlaces de tal forma que el navegador nos muestre todos los documentos destino a la vez. Este sistema, que es en realidad una clase específica de enlace extendido, requiere que el procesador conozca, antes de que se active, todos los documentos implicados.

```
<capítulo1>Tema 1.- La invención de la imprenta</capítulo 1>

<artículos relacionados xml:link=»group»>
        <art xml:link=»articulo» href=»Hipólito Escolar – cap. 4. Xml»>
        <art xml:link=»articulo» href=»David Dahl – cap. 2. xml»>
</artículos relacionados>
```

Este ejemplo nos mostraría, al lado de nuestro texto sobre la aparición de la imprenta, lo que han escrito Hipólito Escolar y David Dahl sobre este tema.

Otro elemento nuevo que introduce XLink son los denominados Punteros X. Este mecanismo permite, por medio de términos de localización, identificar una ubicación concreta dentro de un documento. Además, el Puntero X puede ser matizado con parámetros como número de ocurrencia (del elemento referenciado), tipo de elemento o atributo.

3.2.2. Ejemplos de aplicación documental

A través de los enlaces XLink se podría implementar, en un depósito de referencias bibliográficas en XML, algunas de las opciones de consulta de una base de datos relacional. Utilizando estos enlaces sería posible enlazar todos los documentos de un autor, o todos aquellos que tengan un contenido determinado.

En un proceso posterior, los enlaces podrían utilizarse para indizar los documentos de la misma forma que se usan los vínculos HTML para indizar el Web, pero con una diferencia: los enlaces XLink son más sofisticados y ofrecen más opciones. De esta forma podríamos lograr una indización mucho más representativa.

Si, por último, este mecanismo funcionara junto a otras especificaciones como XSL o XML-QL, las posibilidades de implementación se multiplicarían. Junto a XSL, por ejemplo se podría implementar el concepto de transclusión (generación de nueva información dentro del documento actual), con XML-QL se podrían definir diferentes tipos de comportamiento (query by example) así como gestionar la herencia entre elementos de enlace (para generar y gestionar índices de forma automática).

3.3. XML-QL (eXtensible Markup Language - Query Language)

Una vez convertidos los documentos de la organización a XML, sería necesario contar con un sistema que permita interrogar esa base de datos. El mecanismo utilizado para realizar estas consultas dependerá de cómo se entienda su estructura: estructura tabular (relacional) o arbórea (jerárquica y orientada a objetos).

SQL (Structured Query Language) es un lenguaje asentado y normalizado que se utiliza para la recuperación de datos relacionales. XML presenta la información de forma diferente (pero no opuesta) al modelo relacional: los datos forman un árbol jerárquico de elementos y atributos. Por eso está aumentando el interés en desarrollar un lenguaje que tenga en cuenta las características XML pero que también sea capaz de cubrir las funciones que actualmente desempeña SQL.

Un lenguaje de interrogación compatible con XML facilitaría el proceso de grandes colecciones de datos y la extracción de información relevante. Además incrementaría en gran medida la precisión de las búsquedas, ya que se podrían realizar sobre la estructura jerárquica de los documentos y dispondrían de ciertas facilidades heredadas de las especificaciones XLink y XSL. De hecho, un lenguaje como este supondría incluso una mejora en las actuales tentativas de interrogación a texto completo.

Existen dos especificaciones XML que tratan este tema. Por un lado XQL (eXtensible Query Language), que se encarga exclusivamente de definir los mecanismos y procedimientos necesarios para llevar a cabo la interrogación a la base de datos. Por otro XML-QL, con dos objetivos: la interrogación y el formato de salida de la información. Este apartado se centra en las posibilidades de este último.

XML-QL es un lenguaje que, al igual que SQL, utiliza construcciones de tipo SELECT-WHERE y toma prestadas algunas características de otros lenguajes de interrogación desarrollados recientemente para datos semiestructurados.

Su objetivo es cubrir algunas funciones para las que la especificación XML no se basta por sí misma:

- Extraer datos de grandes colecciones documentales XML.
- Intercambiar datos XML entre diferentes DTDs.
- Integrar datos XML que procedan de distintas fuentes.
- Transportar grandes cantidades de datos XML a los clientes especificados.
- Plantear consultas contra documentos-fuente XML e incluso contra bases de datos cuya estructura sea compatible con el entorno XML.

La extracción, transformación e integración de datos son problemas ya estudiados por los expertos en bases de datos. Sus soluciones a menudo se apoyan en un lenguaje de interrogación de bases de datos relacional como SQL u orientado a objetos como OQL (Object Query Language). Estos lenguajes no pueden aplicarse directamente a XML porque los datos en XML difieren de los que tradicionalmente gestionan las bases de datos relacionales u orientadas a objetos.

Sin embargo los datos XML son muy similares al modelo conocido como «datos semiestructurados». Todos los lenguajes de interrogación tienen un modelo subyacente de datos que formaliza la representación física de los datos reales. Por ejemplo, los lenguajes relacionales operan sobre las relaciones entre los datos y los lenguajes orientados a objetos sobre un tipo característico de datos denominados objetos. Por lo tanto es necesario definir con precisión un modelo de datos también para XML. Lo apropiado según los expertos parece ser aplicar una variante del mencionado modelo de datos semiestructurados. Una vez establecido este modelo se podrían describir características más avanzadas de XML-QL como la transformación o integración de datos.

XML-QL comparte algunas características con XSL, sin embargo no insiste tanto en el formato como en la gestión de datos. Básicamente XML-QL puede expresar consultas y extraer partes de documentos XML para responderlas, puede transformar datos XML a fin de que sean intercambiados entre DTDs distintas y permite integrar datos XML procedentes de fuentes diferentes.

Para ver algunas de estas posibilidades vamos a analizar una DTD y su implementación en un documento XML. La DTD con la siguiente estructura:

```
<?XML version = «1.0»?>
<!DOCTYPE DOCUMENT [
<!ELEMENT libro (autor+, titulo, editor)>
        <!ATTLIST libro fechalibro CDATA #REQUIRED >
        <!ELEMENT articulo (autor+, titulo, fecha?, (versioncortalversionlarga))>
        <!ATTLIST articulo tipoarticulo CDATA #REQUIRED>
        <!ELEMENT editor (nombre, direccion)>
        <!ELEMENT autor (nombre?, apellido)>
]>
```

tendría esta representación formal:

```
<DOCUMENTO>
        <libro fechalibro=»1989»>
                        <titulo>Hypertext hands-on!</titulo>
                        <autor><apellido> Kearsley</apellido></autor>
                        <autor><apellido> Shneiderman</apellido></autor>
                        <editor><nombre> Addison-Wesley</nombre ></editor>
                </libro>
        <libro fechalibro=»1983»>
                        <!— La primera edición en 1976 —>
                        <titulo>Introduction to modern Information Retrieval</titulo>
                        <autor><apellido>Salton</apellido></autor>
                        <editor><nombre>McGraw-Hill</nombre></editor>
        </libro>
</DOCUMENTO>
```

3.3.1. Emparejar datos usando patrones

XML-QL usa el patrón elemento para localizar y emparejar datos en un documento XML. Este ejemplo recupera todos los autores de libros cuyo editor sea Addison-Wesley en el documento XML. El URI (Uniform Resource Identificator) que remite al recurso o recursos de donde se ha extraído la información aparece abajo a la derecha de IN.

```
WHERE
        <libro>
                <editor><nombre>Addison-Wesley</nombre></editor>
                <titulo> $t</titulo>
                <autor> $a</autor>
        </libro> IN «www…/bib.xml»
CONSTRUCT $a
```

Informalmente esta consulta localiza los elementos <libro> en el documento XML: http://www…./bib.xml que tengan al menos un elemento <titulo>, un elemento <autor> y un elemento <editor>; cuyo elemento <nombre> sea igual a: Addison-Wesley. Para cada una de esas respuestas se marca con las variables $t y $a cada uno de los elementos TITULO y AUTOR. El resultado es una lista de autores asociada a la variable $a. (Los nombres de las variables son precedidos por $ para distinguirlos de los literales en el documento XML).

3.3.2. Construcción de datos XML

La consulta anterior recupera una lista de todos los autores (con sus elementos <nombre> y <apellido>) del documento fuente. La siguiente consulta recupera <autor> y <titulo> y los agrupa en un nuevo elemento llamado <resultado>:

```
WHERE
        <libro>
                <editor><nombre>Addison-Wesley</nombre></editor>
                <titulo> $t</titulo>
                <autor> $a</autor>
        </libro> IN «www…/bib.xml»
CONSTRUCT

        <resultado>
                <autor>$a</autor>
                <titulo>$t</titulo>
                </resultado>
```

Si aplicamos esta consulta sobre los datos del ejemplo, la respuesta es:

```
    <resultado>
                    <autor><apellido> Kearsley</apellido></autor>
         <titulo>Hypertext hands-on!</titulo>
             </resultado>
```

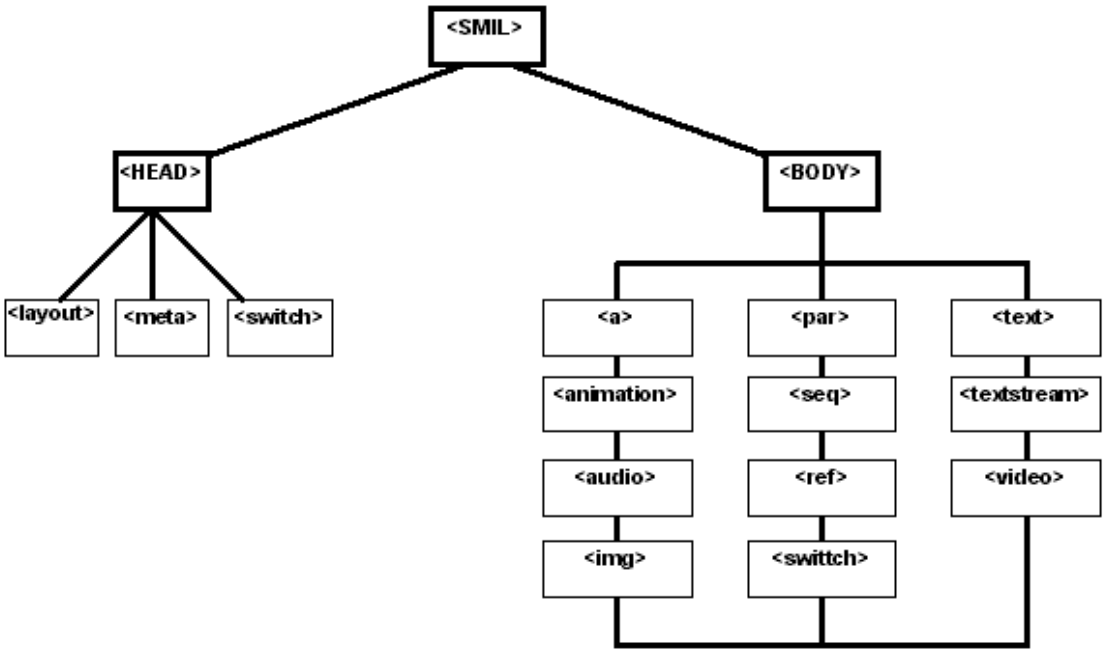
En definitiva XML-QL proporciona un mecanismo para implementar dos funciones importantes: el planteamiento de consultas contra recursos XML y la posibilidad de transformar las respuestas obtenidas en nuevos documentos XML. La principal ventaja de XML dentro del campo de la Documentación consiste en que añade a la potencia del lenguaje SQL la precisión de su modelo de datos basado en XML y por lo tanto en una jerarquía arbórea de objetos y relaciones.

3.4. SMIL (Synchronized Multimedia Integration Language)

En una gran cantidad de sistemas documentales existe la necesidad de gestionar con eficacia información multimedia. Almacenar, editar y distribuir multimedia es, hoy por hoy, tan importante como almacenar, editar o distribuir información textual. SMIL ofrece un marco para la integración y sincronización de objetos multimedia, incluido texto.

SMIL permite la integración de objetos multimedia independientes en una presentación multimedia sincronizada, describiendo el comportamiento temporal de una presentación, describiendo su salida en pantalla, asociando hiperenlaces con los objetos multimedia, etc.

Básicamente, un documento SMIL se compone de un elemento raíz (el elemento SMIL) con un solo atributo (un identificador XML ID). En cuanto a su modelo de contenido, el elemento SMIL cuenta con dos hijos: HEAD y BODY.



3.4.1. Elemento HEAD

Contiene metainformación que no se halla relacionada con el comportamiento temporal de la presentación o las demás funciones de la especificación SMIL. El atributo de «HEAD» es también «*ID*». «HEAD» comprende los siguientes subelementos: «LAYOUT» (determina la posición de los elementos del cuerpo del documento en un «espacio» abstracto de salida), «META» (propiedades de un documento) y «SWITCH» (para anidar elementos LAYOUT). En general «HEAD» puede contener cualquier número de elementos «META» y un elemento «LAYOUT» o «SWITCH».

3.4.2. Elemento BODY

Contiene información relacionada con el comportamiento temporal y los enlaces del documento. Su contenido se podría dividir en elementos de sincronización y elementos asignados a determinados objetos multimedia.

La exactitud de la sincronización entre subelementos depende en gran medida de la aplicación que gestione los documentos SMIL. Supongamos, por ejemplo, que el elemento

«PAR» contiene dos o más tipos de media continuos como audio o vídeo. Si uno de ellos sufre un retraso, un reproductor de audio o vídeo podría mostrar los siguientes comportamientos en cuanto a: sincronización rígida (el reproductor sincroniza los hijos del elemento «PAR» a un reloj común) o sincronización suave (cada hijo del elemento «PAR» se atiene a su propio reloj, que funciona independientemente de los demás).

Mediante el resto de los elementos: «AUDIO», «TEXT», «IMG», «VIDEO», etc. se especifica la inclusión de objetos multimedia en el documento (y la presentación) SMIL. Los objetos se incluyen mediante una referencia (para ello se utiliza los identificadores URI). Básicamente hay dos tipos de objetos: aquellos que tienen una duración intrínseca (continuos): vídeo, audio; y los que no poseen esta característica (discretos): texto, imágenes. La especificación prevé la posibilidad de añadir enlaces a los objetos mencionados.

El reproductor reconoce el tipo de multimedia mediante otras fuentes como el tipo de información, etc. Los autores, sin embargo, por cuestiones de legibilidad, deben asegurarse de que el tipo de multimedia se refleja en el nombre del elemento, cuando se dude acerca del tipo de multimedia al que pertenece un objeto se puede usar el elemento genérico «ref».

La aportación fundamental de la especificación SMIL a la documentación es el concepto de sincronización temporal de objetos (multimedia o textuales, la división es intrascendente en cuanto ambos son objetos) en una presentación. Esta noción podría presentar muchas posibilidades a la hora de diseñar interfaces para sistemas de búsqueda integrados en un sistema documental.

3.5. SOX (Schema for Object Oriented XML)

Esta especificación trata de definir una serie de tipos de datos para XML y una forma normalizada de estructurar los documentos XML de modo que se facilite la orientación a objetos, sobre todo para desarrollo de aplicaciones. SOX aporta a XML:

- tipos de datos intrínsecos,
- un mecanismo de tipificación de datos para definir nuevos tipos,
- un modelo de contenido basado en el concepto de «átomos estructurales»,
- un sistema de herencia entre elementos,
- un mecanismo de nombre localizador tipificado (URL, URN, URI, etc.) y
- la posibilidad de anidar documentación técnica en los documentos.

La orientación a objetos puede ser importante para la documentación, porque el tratamiento que se le da a un objeto (p. ej. un registro bibliográfico o una página Web) implica siempre precisión; porque facilita el desarrollo de aplicaciones automáticas y porque introduce nuevas posibilidades como los conceptos de herencia, métodos, clase, encapsulación, etc.

3.5.1. Ejemplo de aplicación

Dos cuestiones fundamentales que debemos tener en cuenta de SOX con vistas a una posible aplicación documental son: los tipos de datos y la herencia.

El primero de ellos afecta directamente a cuestiones de programación, por lo que no entraremos en este tema. Sin embargo la herencia es un elemento que puede dar mucho juego en un sistema documental. Por ejemplo, el autor de un documento XML podría tener diferentes atributos atendiendo a las diversas normas bibliográficas junto a un elemento subordenado denominado «REFERENCIA». De esta forma, el documento podría heredar de ese elemento el valor de atributo adecuado, según se trate de un tipo de referencia u otra.

En definitiva, las principales aportaciones de SOX tienden a facilitar la inclinación XML hacia la orientación a objetos, las más interesantes desde una perspectiva documental son: la definición de tipos de datos intrínsecos a imitación de muchos lenguajes de programación, la posibilidad de definir nuevos tipos de datos a partir de los intrínsecos, la implementación del concepto de herencia entre elementos (de atributos, de modelos de contenido, etc.).

3.6. WIDL (Web Interface Definition Language)

Esta especificación, desarrollada por la empresa WebMethods, facilita el diseño de aplicaciones Web interactivas. Un ejemplo de este tipo de aplicaciones podría ser un formulario Web. El usuario debe rellenarlo, enviarlo al servidor y esperar que éste le devuelva los resultados en forma de página HTML. Usando WIDL ese tipo de transacciones puede ser completamente automatizado. La definición de una transacción WWW que hace WIDL puede usarse para generar código en, prácticamente, cualquier lenguaje: Java, JavaScript, C++ o Visual Basic, por ejemplo.

WIDL se relaciona con XML de dos formas: en primer lugar lo usa como código para describir transacciones WWW, por otro lado, la tecnología WIDL se puede aplicar también para automatizar el acceso WWW a recursos XML.

Para ver someramente algunas de las posibilidades de WIDL nos centraremos en el siguiente formulario HTML que permitiría a un usuario especificar un valor límite para el tamaño de disco duro.

```
<HTML>
        <HEAD>
        <TITLE>Tamaño máximo</TITLE>
        </HEAD>
        <BODY>
            <H1>Selección de tamaño de disco duro</H1>
            <CENTER>
            <FORM ACTION=«../scripts/db.py» METHOD=post>
                <INPUT Type=«Submit» Value=«Select a minimum hard disk size»>
                <INPUT Type=«String» Name=«Hdcapacity» Value=«10000»>
            </FORM>
            </CENTER>
        </BODY>
</HTML>
```

Si se tiene la intención de automatizar este formulario, lo primero que hay que hacer es analizar su contenido. El Web Automation Toolkit de Webmethods (público, al igual que WIDL, en http://www.webmethods.com) recupera de su URL el código HTML presentado como ejemplo y aísla las partes del mismo que requieran interacción. Estas partes son «encuadernadas» en los elementos «BINDING» (en este caso de entrada). A partir del formulario que se quiere automatizar se empieza a crear un documento WIDL, en el que aparecería el siguiente elemento de entrada:

```
<BINDING NAME=«HdInput» TYPE=«ÏNPUT»>
        <VARIABLE
                NAME=«HdCapacity»
                TYPE=«String»
                FORMNAME=«HdCapacity»
                USAGE=«DEFAULT»
                COMMENT=««»>
        </VARIABLE>
</BINDING>
```

En realidad el parser WIDL divide las páginas HTML que analizan en sus componentes fundamentales. Esos componentes se hallan referenciados mediante una sintaxis que usa el modelo de objetos JavaScript (actualmente estas referencias se están implementando en DOM).

Se ha dividido el documento HTML en sus componentes básicos y esta información ha sido trasladada a un elemento que se ocupa de la entrada: <BINDING NAME=«HdInput» TYPE=«INPUT»>. Pero en cualquier transacción hay una parte de entrada y otra de salida, así que a partir de la salida del formulario HTML (una vez activado), el parser WIDL tendría que generar un elemento de salida análogo (descomponiendo el código en sus elementos fundamentales como hizo con la entrada), por ejemplo:

```
<BINDING NAME=«HdOutput» TYPE=«OUTPUT»>
        <VARIABLE
                NAME=«title»
                TYPE=«String»
                REFERENCE=«doc.title[0].text
                USAGE=«DEFAULT»
        </VARIABLE>
        <VARIABLE
                NAME=«doctext»
                TYPE=«String»
                REFERENCE=«doc.text
                USAGE=«DEFAULT»
        </VARIABLE>
        <VARIABLE
                NAME=«paragraphs»
                TYPE=«String»
                REFERENCE=«doc.p[].text
                USAGE=«DEFAULT»
        </VARIABLE>
        <VARIABLE
                NAME=«table0»
                TYPE=«String [] []»
                REFERENCE=«doc.table[0].tr[].th _td [].text»
                USAGE=«DEFAULT»
        </VARIABLE>
</BINDING>
```

Se puede observar cómo en este elemento se halla especificado todo un documento HTML. La última variable se refiere a una tabla que es la que devolverá al usuario la información que ha requerido. El parser WIDL generará, con toda la información que se ha recogido sobre la transacción HTML: un fichero «readme», un documento WIDL, un programa Java independiente, un applet Java y un documento HTML que contenga dicho applet.

De esta forma se consigue organizar la información de tal modo que se facilita la automatización de una sesión WWW interactiva (con lo que esto implica para los sistemas documentales).

3.6.1. Ejemplo de aplicación

WIDL es una herramienta muy útil para implementar interfaces interactivas para cualquier función dentro de un sistema documental: peticiones de usuarios, altas de usuarios, intercambio de registros entre bibliotecas, consultas a bases de datos distribuidas o, simplemente, una búsqueda básica de recursos.

4. Conclusiones

Todas las posibilidades de trabajo que ofrecen las especificaciones XML pueden ser aplicadas a cualquier sistema de información, cubriendo un gran porcentaje de las necesidades de profesionales y usuarios.

En estas páginas se han propuesto dos vías de solución a los problemas documentales tanto en el Web como en entornos documentales específicos: por un lado la automatización de procesos (desarrollando aplicaciones con XML que puedan cubrir determinadas funcionalidades); por otro la estructuración (integración de formatos en una arquitectura orientada a objetos). Ambos mecanismos permiten mejorar la gestión de un centro en cualquiera de sus facetas (creación, edición, almacenamiento, recuperación…).

A lo largo del trabajo se han analizado algunas de las especificaciones XML más conocidas y, sobre todo, más aplicables al campo de la documentación electrónica. Lo que todas las especificaciones tienen en común es: la proximidad al paradigma de la orientación a objetos y el tipo de estructura arbórea con la que gestionan la información (no importa que el árbol contenga elementos textuales, elementos de formato, multimedia o elementos de una interfaz). En opinión de los autores estas dos características resultarán vitales a corto-medio plazo para cualquier sistema que pretenda gestionar documentación electrónica.

Bibliografía

DOM (Document Object Model): http://www.w3.org/DOM/

EAD (Encoded Archival Description): http://www. loc.gov/ead/

IMS Metadata Specification: http://www.imsproject.org

MathML (Mathematical Markup Language): http://www.w3.org/Math/

MCF (Meta Content Framework): http://www.w3.org/TR/NOTE-MCF-XML.html

OFX (Open Financial Exchange): http://www.ofx.net

OTP (Open Trading Protocol): http://www.otp.org

Página oficial del W3C para XML: http://www.w3.org/XML/Activity

PGML (Precision Graphics Markup Language): http://www.w3.org/TR/1998/NOTE-PGML

RDF (Resource Description Framework): http://www. w3c.org/RDF/

SGML (Standard Generalized Markup Language): http://www.sgmlopen.org/

SGML / XML: http://www.oasis-open.org/cover/xml.html

SMIL (Synchronized Multimedia Integration Language): http://www.w3.org/TR/REC-smil/

SOX (Schema for Object Oriented XML): http://www.w3.org/TR/NOTE-SOX/

UML (Unified Modeling Language): http://www.rational.com/uml/index.jtmpl

UML-Xchange: http://www.cam.org/~nrivard/uml/umlxchng.html

VML (Vector Markup Language): http://www.w3.org/TR/NOTE-VML

WML (Wireless Markup Language): http://www.wapforum.org/

XLF (eXtensible Log Format): http://www.docuverse.com/xlf/index.html

XLink (XML Linking Language): http://www.w3.org/TR/WD-xml-link

XML (eXtensible Markup Language): http://www.w3.org/XML/

XML/EDI: http://www.xmledi.net

XML-QL: http://www.w3.org/TR/NOTE-xml-ql/

XSL (eXtensible Style Language): http://www.w3.org/TR/WD-xsl/