Integración, navegación, presentación: experiencias utilizando XML

Oscar Díaz, Jon Iturrioz, Felipe Ibáñez

Grupo EKIN
Departamento de Lenguajes y Sistemas Informáticos
Universidad del País Vasco / Euskal Herriko Unibertsitatea
Apartado 649
20.080 San Sebastián/Donostia
Tfno: 943 44 80 00
Fax: 943 21 93 06
e-mail: jipdigao | jibitsaj | jibibanf @si.ehu.es

Resumen

La creciente necesidad de integrar aplicaciones, tanto dentro de la propia organización como con organizaciones asociadas, ha promovido el estándar XML. Este estándar permite describir la estructura de los documentos, con independencia, tanto de su presentación, como del soporte donde se asienten sus datos. Alrededor de XML están surgiendo diferentes lenguajes que se centran en algún tratamiento específico del documento. Este trabajo presenta nuestra experiencia en la utilización de XML para la realización de aplicaciones hipermedia. Las aplicaciones hipermedia se caracterizan por organizar los datos en grafos de nodos, y donde el acceso se realiza por medio de enlaces entre los nodos. En la concepción y desarrollo de estas aplicaciones, se suelen distinguir los siguientes esquemas: esquema conceptual, esquema de derivación, esquema de navegación y esquema de presentación. En nuestro caso utilizamos XML para describir el esquema conceptual, XML-QL para el esquema de derivación, un lenguaje *ad-hoc* inspirado en XQL para el esquema de navegación, y finalmente, XSLT para el esquema de presentación. Este trabajo ilustra la utilización de estos lenguajes para la realización de aplicaciones hipermedia.

Palabras clave: XML, aplicaciones hipermedia, arquitectura de tres niveles.

1. Introducción

Las inversiones realizadas en sistemas de información han permitido una gestión más ágil de los procesos de negocio, especialmente la de aquellos soportados mediante aplicaciones transaccionales o aplicaciones OLAP. Sin embargo, esta automatización se ha centrado, primero en procesos internos a la organización, y segundo, en datos de tipo tabular. El primer aspecto conlleva que las relaciones con agentes externos a la empresa se sigan gestionando mayoritariamente a través de impresos (p. ej. pedidos, facturas, etc), cuyos datos son luego introducidos manualmente en el sistema. El segundo aspecto soslaya que un ingente volumen de información no se almacena en bases de datos. De hecho, se estima que el 90% de la información encuentra soporte en páginas HTML, documentos WORD, hojas de cálculo, o simples ficheros. Esta situación no se debe únicamente a la incapacidad

de los Sistemas de Gestión de Bases de Datos (SGBD) tradicionales para dar soporte a tipos de datos y de manipulación diferentes a los meramente tabulares. Los SGBD también imponen una disciplina en la utilización que puede resultar excesivamente restrictiva para gestionar determinados tipo de datos. Por ello, es previsible que las organizaciones sigan confiando sus datos a soportes heterogéneos, eligiendo en cada caso el soporte que mejor se adapte a sus necesidades de manipulación, expresividad, seguridad y capacidad para compartir.

La situación actual es por tanto, la de un archipiélago donde sólo determinados aspectos dentro de la empresa han sido automatizados (las islas), y el trasiego entre "islas" requiere de procesos manuales. XML y sus lenguajes aledaños (XSLT, XML-QL, XQL, etc) han sido propuestos con esta finalidad. XML ofrece la flexibilidad y ductilidad necesaria para adaptar e integrar los distintos impresos al formato esperado por la aplicación receptora.

El objetivo es distinguir entre tres aspectos: contenido, organización y presentación de los datos. La gestión del contenido es el afán tradicional de los SGBD. La gestión eficiente de este contenido ha conllevado una disposición de los datos en estructuras sencillas como la representación tabular. Si bien esta estructura facilita el tratamiento informático, resultan excesivamente *"plana"* para reflejar el tipo de impresos que maneja una organización. XML permite describir más fielmente esta estructura no tabular de los impresos tradicionales. XML es un lenguaje sencillo, que mediante la utilización de *marcas* permite al usuario definir la estructura y contenido de sus datos. Estas características lo hacen adecuado para ser utilizado como *linguae franca* para la integración de aplicaciones y fuentes de datos heterogéneas. Por último, la presentación se desliga de la estructura utilizando lenguajes como XSLT, evitando la descripción conjunta de contenido, estructura y presentación que se daba en HTML. Esta clara separación entre estos tres aspectos permite que el mismo contenido sea estructurado de distintas formas, reflejando así los distintos tipos de impresos dentro de una organización donde los mismos datos aparecen dispuestos de maneras diferentes. Asimismo es viable que la misma estructura siga diferentes patrones de presentación, adaptándose así a las características del terminal receptor o a las peculiaridades del usuario final.

Este trabajo presenta nuestra experiencia en la utilización de la saga XML para separar estos tres aspectos (contenido, organización y presentación) a los que se le añade la navegación. La navegación es la forma de trabajo predominante en Internet, lo que ha promovido el interés por las denominadas aplicaciones hipermedia. Estas aplicaciones se caracterizan por organizar los datos (potencialmente multimedia) en grafos de nodos, y donde el acceso se realiza por medio de enlaces entre los nodos. De esta manera, el usuario obtiene la información *"navegando"* entre los nodos, utilizando la denominada metáfora *"point and click"* [7].

En la concepción y desarrollo de estas aplicaciones, se suelen distinguir los siguientes esquemas: esquema conceptual, esquema de derivación, esquema de navegación y esquema de presentación. En nuestro caso utilizamos XML para describir el esquema conceptual, XML-QL para el esquema de derivación, un lenguaje *ad-hoc* inspirado en XQL para el esquema de navegación, y finalmente, XSLT para el esquema de presentación.

Para la presentación de este trabajo se ha seguido la siguiente estructura. La siguiente sección aborda la integración de múltiples fuentes de datos. La secciones 3 y 4 tratan el esquema de navegación y el esquema de presentación, respectivamente. La arquitectura del enfoque se presenta en la sección 5. La última sección se dedica a las

conclusiones.

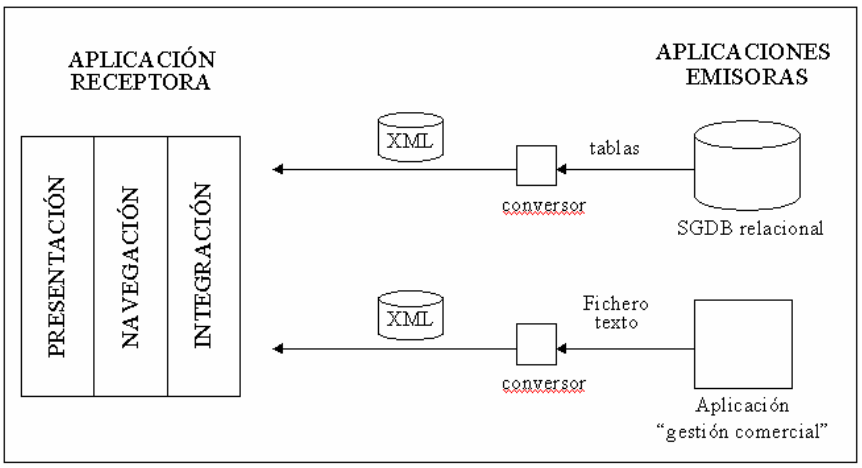

Figura 1: Una aplicación integra documentos procedentes de distintas fuentes.

2. Integración de múltiples fuentes de datos

La figura 1 muestra el enfoque seguido. Tenemos una aplicación (en nuestro caso, una aplicación hipermedia) cuyos datos provienen de un conjunto de aplicaciones. Estos datos tienen una estructura y origen heterogéneo, y son obtenidos bajo demanda, es decir, seguimos un enfoque *"pull"*. Al trabajar con distintas fuentes de datos es aplicable la arquitectura de los sistemas federados de bases de datos (SFBD) , donde se distinguen los siguientes esquemas [10,3]:

- Esquema local. Datos y estructura tal y como los facilita la aplicación emisora. En el caso de un SGBD relacional, el esquema será un conjunto de tablas. Si la aplicación ya facilita XML, el esquema local corresponderá al *Document Type Definition* (DTD) asociado. Finalmente si se trata de un fichero de texto, el esquema local será la estructura del registro.
- Esquema componente. Representación del esquema local en términos del modelo canónico de datos. En nuestro caso, escogemos XML como lenguaje común. Si la aplicación emisora no utiliza este modelo, será necesario un conversor para pasar el esquema local a XML.
- Esquema federado. Esquema resultante de la integración de varios esquemas exportados. Para ello, se utilizan mecanismos de vistas. Para el caso de documentos XML, estos mecanismos todavía están en su infancia [1]. Una posibilidad es utilizar XSLT[2]. A partir de un documento XML, XSLT genera todo tipo de texto. Normalmente el texto generado es HTML y por ello se utilizan para indicar las hojas de estilo (*"stylesheets"*) empleadas en la presentación del documento XML. En cualquier caso, XSLT es un mecanismo muy rudimentario que no ha sido concebido para soportar la definición de vistas. Por ello, y siendo conscientes de que no se trata de un estándar, hemos preferido utilizar XML-QL [6]. Esta propuesta de lenguaje de interrogación para XML ha sido presentada al W3C en 1998 y una implementación esta

disponible para Unix.

Las similitudes con los sistemas de bases de datos federados son por tanto, evidentes. Sin embargo, las aplicaciones hipermedia presentan una serie de peculiaridades, a saber: 1) son mayoritariamente de "sólo lectura", es decir, sólo se van a consultar los datos, no hay modificaciones, 2) los datos utilizados son estables, es decir, no sufren grandes modificaciones por parte de otras aplicaciones (p. ej. el expediente de un alumno, los establecimiento hoteleros de una ciudad tienden a ser fijos una vez introducidos). Estas razones nos mueven a supeditar la definición del esquema federado a la aplicación, es decir, la concepción del esquema federado se realiza como una etapa mas dentro del desarrollo de la aplicación hipermedia, y su definición forma parte de la descripción de la aplicación hipermedia (véase figura 1). Seguimos pues un enfoque federado débilmente acoplado donde los problemas de heterogeneidad semántica son resueltos por la persona que realiza la aplicación.

2.1 Un ejemplo: integración de los datos de una Facultad

```
<?xml version='1.0'encoding='UTF-8'?>
<! ELEMENT GuiaDocente (Asignatura+)>
<! ELEMENT Asignatura (NomAsi,Ciclo,
        Lengua, Objetivo,Contenido,Profesor+,)>
<! ELEMENT NomAsi PCDATA>
<! ELEMENT Ciclo PCDATA>
<! ELEMENT Lengua PCDATA>
<! ELEMENT Objetivo PCDATA>
<! ELEMENT Contenido (Tema+)>
<! ELEMENT Tema (Numero,Titulo,Objetivo,Dificultad)>
<! ELEMENT Numero PCDATA>
<! ELEMENT Titulo PCDATA>
<! ELEMENT Profesor (Nombre,Apellido1,Apellido2)>
<! ELEMENT Nombre PCDATA>
<! ELEMENT Apellido1 PCDATA>
<! ELEMENT Apellido2 PCDATA>
```

```
<?xml version='1.0'encoding='UTF-8'?>
<! ELEMENT Test (Asignatura+)>
<! ELEMENT Asignatura (Nombre,Parte+)>
<! ELEMENT Nombre PCDATA>
<! ELEMENT Parte(Numero,Dificultad,Tema+)>
<! ELEMENT Tema(Numero,Ejercicio+)>
<! ELEMENT Ejercicio
(Enunciado,RespuestaCorrecta,Opcion+)>
<! ELEMENT Opcion (Numero,Respuesta)>
.....
-----------------------------------
create table Alumno(
     dni number primary key,
     nomAlu varchar(30),
     direccion varchar(40) ....)
create table Matricula(
   elAlumno number not null references Alumno (dni),
   laAsignatura varchar(30)
      not null references Asignatura(nomAsi),
   convocatoria number,
   primary key(elAlumno,laAsignatura))
```

Figura 2: Esquema relacional, y DTD asociados a las fuentes de datos del ejemplo.

Como ejemplo que iremos desarrollando a lo largo del trabajo, hemos tomado la gestión de los datos en una Facultad. La secretaría de la Facultad confecciona la guía docente donde se da información detallada sobre fechas de exámenes, contenidos de asignaturas, horarios, etc. Estos datos se encuentran en un documento XML: *"guiaDocente.xml"*. También es responsabilidad de la secretaría confeccionar las listas de clase donde se relacionan los alumnos junto con la convocatoria en la que se encuentran para cada asignatura. Estos datos están almacenados en un SGBD relacional. Respecto a los profesores, generan apuntes, ejercicios de test, exámenes, etc. En concreto los tests, se guardan como documentos XML: *"test.xml"*.

```
WHERE
<asignatura>
    <nomAsi> $na </>
    <ciclo> $ci </>
```

```
        <lengua> $le </>
        <objetivo> $ob </>
</asignatura> CONTENT AS $asigGuia_c IN "guiaDocente.xml"
<asignatura>
        <nombre> $na </>
 </asignatura> CONTENT AS $asigTest_c IN "test.xml"
<matrícula>
        <laAsignatura> $na </>
  </matrícula> CONTENT AS $asigMatrícula_c IN "matrícula.xml"
CONSTRUCT
<asignatura>
        <nomAsi> $na </>
        <ciclo> $ci </>
        <lengua> $le </>
        <objetivo> $ob </>
        WHERE <alumno> $al </> IN $asigMatrícula_c
        CONSTRUCT <alumno> $al </>
        WHERE <profesor> $pr </> IN $asigGuia_c
        CONSTRUCT <profesor> $pr </>
        WHERE
                <tema>
                        <titulo> $ti </>
                        <objetivo> $obj</>
                        <dificultad> $di </>
                        <numero> $nu</>
                </tema> IN $asigGuia_c
                 <tema>
                        <numero> $nu</>
                </tema> CONTENT AS $ejercicio_c IN $asigTest_c
        CONSTRUCT
                <tema>
                         <titulo> $ti </>
                        <objetivo> $obj </>
                        <dificultad> $di </>
                        WHERE <ejercicio> $ej </> IN $ejercicio_c
                        CONSTRUCT <ejercicio> $ej </>
                </tema>
</asignatura>
```

Figura 3: Documento de integración, utilizando XML-QL.

Los esquemas correspondientes a esta aplicación son:

- esquema local. Tres van a ser las fuentes de datos: la base de datos de matrícula, el documento de test, y el documento con la guía docente. La figura 2 muestra la definición de las tablas y los DTD correspondientes a estas fuentes de datos.
- esquema componente. Al utilizar XML como lenguaje canónico, es necesario transformar a este formato los datos residentes en la base de datos. Para ello, muchos SGBD disponen de paquetes que realizan esta conversión. En nuestro caso, estamos interesados en obtener un documento XML donde para cada asignatura se obtiene una lista de alumnos, y donde de cada alumno se extrae el *nomAlu*, el *apellido1*, el *apellido2*, y la *convocatoria*.
- esquema federado. Para filtrar e integrar los distintos documentos XML utilizamos XML-QL [6]. En nuestro ejemplo, los datos sobre las asignaturas se encuentran desperdigados por:
    - la guía docente, de donde se obtiene el nombre de la asignatura, el objetivo, el ciclo y la lengua en la que se imparte. Además se recupera también el título de los temas que componen la asignatura,
    - el fichero de los ejercicios de test, de donde se extrae para cada uno de los temas de la asignatura, el objetivo, la dificultad y la colección de ejercicios,

- la base de datos de secretaría, de donde se obtienen los alumnos (su nombre completo junto la convocatoria para esta asignatura).

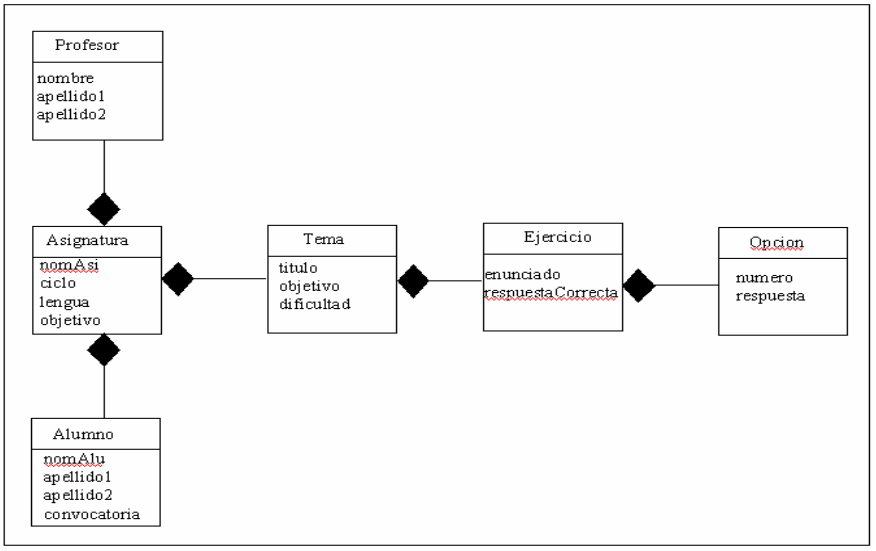

Figura 4: Esquema federado. Notación UML.

La figura 4 muestra el modelo UML correspondiente al documento extraído. La figura 3 muestra la pregunta XML-QL que obtiene este documento a partir de los esquemas componentes.

XML-QL extrae los datos utilizando *"pattern-maching"*. Las variables, cuyos identificadores empiezan por *"$"*, son instanciadas en la parte *"where"*, y el nuevo documento se va construyendo en la parte *"construct"*. Si el nuevo documento no es plano, sino que tiene elementos compuestos, entonces hay que recurrir a preguntas imbricadas. Estas preguntas imbricadas trabajan sobre una parte de los documentos fuente. Esta parte se selecciona con la cláusula *"content as"*, donde la variable asociada aloja el sub-documento comprendido entre las dos etiquetas. Así en el ejemplo
<asignatura> <nombre> $na </nombre> </asignatura> CONTENT AS $asigTest_c IN "test.xml"
la variable *"$asigTest_c"* corresponde al sub-documento correspondiente a la asignatura cuyo nombre es el contenido de *"$na"*.

Resumiendo, durante esta fase el diseñador confecciona el documento XML deseado a partir de los documentos de las aplicaciones emisoras. El resultado se recoge en el **documento de integración**

3. El esquema de navegación

Este esquema describe los enlaces que conformaran la aplicación hipermedia. Estos enlaces se sustentan en el esquema federado, pero sin modificarlo. Al igual que en otros enfoques, distinguimos dos tipos de enlaces: contextuales y no contextuales [4,9]. Los primeros se realizan a través de las asociaciones indicadas en el esquema federado, desde un objeto a otro objeto relacionado. En cambio, los enlaces no contextuales son aquellos que están disponibles desde cualquier punto de la navegación, independientemente del contexto en el

que nos encontremos.

La definición de un enlace se realiza a través de las siguientes propiedades:

- **title**. Constituye la etiqueta del enlace.
- **from**. Indica el tipo de (sub)documento fuente. Su identificación se realiza utilizando una notación de caminos donde se ha ampliado la propuesta realizada para el XQL [8]. Por ejemplo: *//Asignatura/Tema*, identifica a los documentos *"Tema"* dentro de un documento *"Asignatura"*. Cuando la navegación es no contextual, el *"from"* toma el valor "*" para denotar que independientemente de cual fuera el origen, siempre se podrá alcanzar el destino asociado. Asimismo, para denotar los enlaces de arranque es decir, aquellos disponibles en la página inicial, se utiliza el valor *"home"*.
- **to**. Identifica el conjunto de sub-documentos destino de la navegación. Al igual que en el caso anterior, se utiliza notación de caminos para su identificación. Sin embargo aquí, el camino puede ser absoluto o relativo al documento origen. Veamos varios ejemplos:
  - *Tema/Ejercicio*: es un camino relativo. Dado un origen *//Asignatura*, identifica al conjunto de ejercicios para la asignatura en la que nos encontremos.
  - *//Asignatura/Alumno[nombre=$from/Profesor/nombre]*: es un camino absoluto. El camino origen se guarda en la variable *"$from"*. Dado un origen //Asignatura, identifica a los alumnos que se llaman igual que alguno de los profesores de la asignatura identificada por el origen.
  - *../../Profesor*: es un camino relativo. Dado un origen *//Asignatura/Tema/Ejercicio*, el camino sube dos niveles, hasta *Asignatura*, y de allí obtiene los profesores de esta asignatura.

. .

- **indexing**, presenta un índice sobre los elementos de destino. Su valor es la propiedad sobre la que se crea el índice.
- **indexingWhere**, indica si el índice se muestra asociado al enlace (valor *"tooltip"*), en la misma página que el elemento de origen (valor *"embed"*) o en una página separada (valor *"detach"*)
- **showing**, indica si los elementos de destino se muestran cada uno en una página diferente (valor *"one"*) o todos juntos en la misma página (valor *"all"*)
- **showingWhere**, indica si los elementos de destino se muestran en la misma página que el elemento de origen (valor *"embed"*) o en una página separada (valor *"detach"*)
- **sorting**, indica la propiedad que servirá para ordenar los elementos de destino
- **when**, indica el momento de realizar la navegación del elemento origen a los elementos destino. Sus valores pueden ser *"onLoad"* y *"onRequest"* dependiendo de que la navegación se inicie tan pronto se cargue el elemento origen, o cuando así lo indique el usuario, respectivamente.

. .

El resultado de esta fase es el **documento de navegación**.

3.1 Un ejemplo: navegación a través de los datos de la asignatura

La figura 5 muestra los distintos enlaces de navegación dentro del esquema federado mostrado en la figura 4. Inicialmente se ofrecen tres índices sobre las asignaturas: sobre ciclo, sobre lengua, y sobre el nombre de la asignatura, este último disponible desde cualquier punto de la navegación. Una vez seleccionada una asignatura por el usuario, se puede navegar hacia: los profesores, los alumnos o los temas de la asignatura escogida

(representados por las tres flechas de salida en la figura 5). Recuérdese que desde cualquiera de ellos se puede volver al índice sobre el nombre de las asignaturas. Si optamos por los temas, y una vez escogido uno de ellos, el usuario puede navegar a los ejercicios propuestos para este tema, y desde allí, volver, bien a los temas de esta misma asignatura o bien, a todos los temas de cualquier asignatura. En este último caso se le ofrece un índice por título del tema. La navegación del ejercicio a sus opciones es "embed" y "onLoad" es decir, las opciones se presentan conjunta y simultáneamente con los ejercicios (en negrita en la figura 6). Todos estos aspectos son recogidos en el documento de navegación que se muestra en la figura 6.

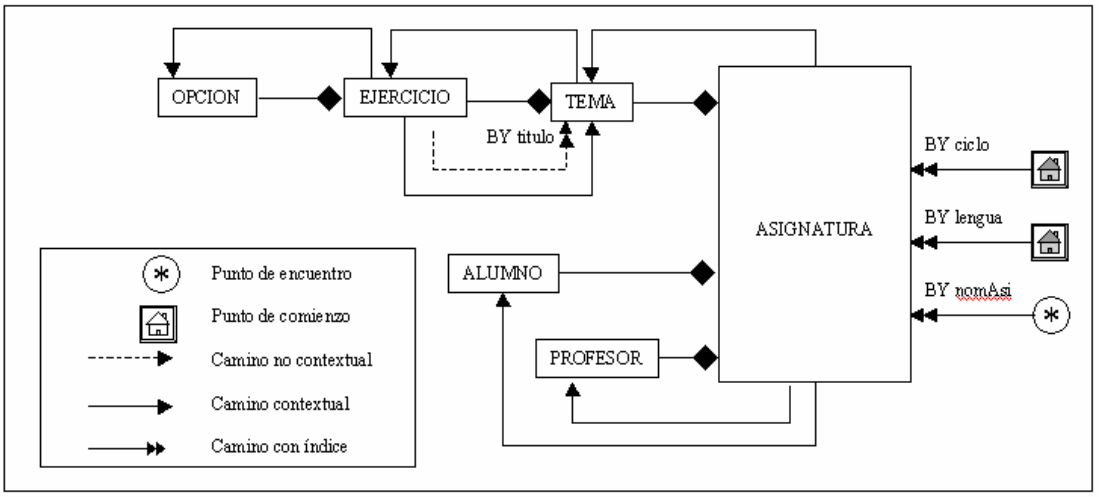

Figura 5: Esquema de navegación.

```
 <?xml version='1.0'?>
<NAVEGACION>
<LINK
      title="AsignaturaPorNombre" from="*" to="//Asignatura"
       indexing="nomAsi" indexingWhere="detach" when="onDemand" />
<LINK
      title='AsignaturaPorCiclo' from='home' to='//Asignatura'
       indexing='ciclo' indexingWhere="detach" showing="one" showingWhere="detach" when="onDemand" />
<LINK
      title='AsignaturaPorLengua' from='home' to='//Asignatura'
       indexing='lengua' indexingWhere="detach" showing="one" showingWhere="detach" when="onDemand" />
<LINK
      title='aTema' from='//Asignatura' to='/Tema'
       indexing="titulo" indexingWhere="tooltip"showing="one" showingWhere="detach" when="onDemand" />
<LINK
      title='aEjercicio' from='//Asignatura/Tema' to='/Ejercicio'
       showing="all" showingWhere="embed" when="onLoad" />
<LINK
      title='aEjercicio' from='//Asignatura/Tema/Ejercicio' to='/Opcion'
      showing="all" showingWhere="embed" when="onLoad" />
<LINK
      title='volverAlTema' from='//Asignatura/Tema/Ejercicio' to='../'
       showing="all" showingWhere="detach" when="onDemand" />
```

```
<LINK
      title='volverATodosLosTemas' from='//Asignatura/Tema/Ejercicio' to='//Asignatura/Tema'
       indexing="titulo" indexingWhere="tooltip" showing="one" showingWhere= "detach" when="onDemand" />
.............
</NAVEGACION>
```

Figura 6: Documento de navegación.

4. El esquema de presentación

Este esquema describe la presentación de cada uno de los elementos del esquema federado. El lenguaje utilizado es XSLT. XSLT utiliza reglas (*patrón*, *acción*) (denominadas *"template rules"*), donde el patrón identifica un nodo del documento XML que en caso de encontrarse en el documento de entrada, conlleva la realización de la acción. En este caso, la acción es una secuencia de instrucciones HTML que indica cómo se visualiza el nodo instanciado en la parte *"patrón"*.

El documento de navegación da la pauta sobre los nodos a presentar: aquellos que aparecen como eslabón terminal en el camino especificado en el atributo *"to"* de algún enlace (p. ej. del camino *"//Asignatura/Tema/Ejercicio"* se desprende que es necesario presentar los nodos correspondientes a ejercicios).

El resultado de esta etapa es el **documento de presentación** Por tanto, una aplicación se compone de 1) un documento de integración, 2) un documento de navegación, y 3) un documento de presentación. Todos ellos se aglutinan en un **documento de la aplicación**. Es conveniente observar que todos estos documentos están descritos en XML. La especificación totalmente declarativa de los distintos aspectos de la aplicación, facilita tanto su desarrollo como su posterior mantenimiento. Incluir nuevos documentos o definir nuevos enlaces se limita a una rápida labor de edición.

4.1 Un ejemplo: presentación de los datos de una asignatura

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl='http://www.w3.org/1999/XSL/Transform version='1.0'>
<xsl:template match='Asignatura'>
  Titulo : <I> <xsl:value -of select='nomAsi'/> </I>
</xsl:template>

<xsl:template match="Tema'>
  <B> Titulo : <xsl:value-of select='Titulo'/> </B>
  <B> Dificultad : <xsl:value-of select='Dificultad'/> </B>
  <B> Objetivos : <xsl:value-of select='Objetivos'/> </B>
</xsl:template>

<xsl:template match='Ejercicio'>
  <B> Enunciado : <xsl:value-of select='Enunciado'/> </B>
  <B> Opciones : </B>
  <SELECT>
    <xsl:attribute name='NAME'>
      <xsl:value-of select='Numero'/>
    </xsl:attribute>
    <xsl:for-each select='Opcion'>
      <OPTION>
        <xsl:attribute name='VALUE'>
          <xsl:value-of select='Numero'/>
        </xsl:attribute>
        <xsl:value -of select='Respuesta'/>
      </OPTION>
    </xsl:for-each>
  </SELECT>
</xsl:template>
</xsl:stylesheet>
```

Figura 7: Documento de presentación.

A partir del documento de navegación que se muestra en la figura 6, se extrae que los nodos a presentar son: *"Asignatura", "Tema"* y *"Ejercicio*". Para cada uno de ellos definimos una regla (*patrón, acción*) utilizando XSLT. La figura 7 muestra el documento de presentación que engloba estas tres reglas. Una regla esta asociada a la etiqueta *<xsl:template>*. El patrón corresponde al atributo *"match"*. La parte acción corresponde al código HTML donde se puede hacer referencia a los atributos del nodo localizado en el patrón mediante la instrucción *<xsl:value-of select='nombreDelElemento'/>*. Así la primera regla, para cada nodo de tipo *"Asignatura"*, presenta el texto *"Título :"* seguido del valor del atributo *"nomAsi"* asociado a esta asignatura.

```
<?xml version='1.0'?>
<aplicacion >
  <docIntegracion> integracionFacultad.xml </docIntegracion>
  <docNavegacion> navegacionFacultad.xml </docNavegacion>
  <docPresentacion> presentacionFacultad.xsl </docPresentacion>
</aplicacion>
```

Figura 8: Documento de aplicación.

Ya para terminar, la figura 8 muestra el documento de aplicación para el ejemplo que hemos ido desarrollando. Los *documentos "integracionFacultad.xml"*,

"navegacionFacultad.xml" y *"presentacionFacultad.xsl"* corresponden a los documentos que hemos ido presentando en las secciones anteriores.

5. Arquitectura

Con los servidores WEB las peticiones del cliente viajan hasta el servidor que devuelve la página HTML a visualizar. El diálogo se hace en términos de presentación: la página HTML. Si estas páginas se limitan a la presentación, cada página puede requerir una nueva conexión al servidor. Esto ralentiza todo el proceso y está en la génesis del término "*World Wide Wait*". Sin embargo, los nuevos lenguajes de "*scripting*" y XML hacen que las páginas no se tengan que limitar a la presentación. Si bien la página sigue siendo el envoltorio, ahora puede embeber programas y datos. Esto permite que la unidad de transferencia no tiene necesariamente que coincidir con la unidad de interacción. La página transferida puede contener los datos y los procesos que permitan generar **localmente** otras páginas, evitando así nuevas conexiones con el servidor.

El dominio que hemos escogido nos permite beneficiarnos de estos avances ya que conocemos *a priori* los posibles datos y la navegación que va a efectuar el usuario. Esta información esta descrita **declarativamente** en XML, lo que permite su acceso a través de un "*script*". En un caso extremo, la aplicación requeriría una única conexión con el servidor, recuperando los tres documentos (navegación, presentación e integración) de una sola tacada. Estos documentos serían luego procesados localmente dando como resultado distintas páginas HTML.

Como no puede ser de otro modo, la arquitectura seguida es de tres niveles. La figura 9 ilustra la interacción entre los distintos niveles, a saber:

1. el cliente solicita la pagina de arranque de la aplicación.
2. el servidor de aplicaciones recibe la petición y determina que se trata de una aplicación XML, recuperando el *documento de aplicación* asociado. De aquí obtiene el *documento de integración* que se lo envía para su procesamiento al integrador.
3. el integrador, determina primero la fuente de los datos.
   - si la fuente es un fichero XML, se solicita directamente los datos al servidor de ficheros.
   - si la fuente es un SGBD relacional, se solicita los datos a través de una pregunta SQL al conversor, en este caso "*sql2xml"*
   - en el caso de fuentes con otro modelo de datos, se realizaría la pregunta al conversor correspondiente.
4. el conversor recupera los datos en el modelo local y los convierte a XML. La respuesta es enviada al integrador.
5. consultadas todas las fuentes de datos, el integrador envía los documentos XML recopilados junto con el documento de integración, al procesador de XML-QL.
6. el procesador de XML-QL devuelve el esquema federado al integrador, que a su vez, lo reenvía al servidor.

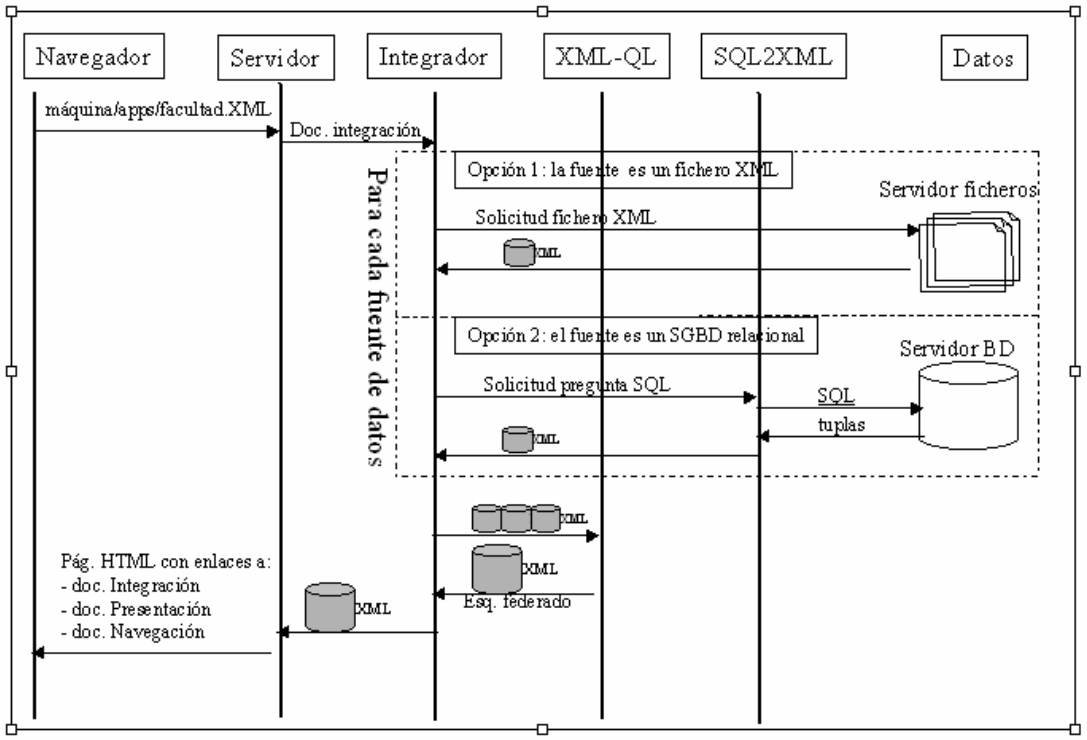

Figura 9: Arquitectura.

7. finalmente, el servidor responde a la petición del cliente, enviándole la página de arranque que contiene el documento de integración, el documento de presentación y el documento de navegación.
8. una vez en el cliente, la aplicación se ejecuta localmente, sin necesidad de recurrir al servidor.

6. Conclusión

Este trabajo ilustra la utilización de XML para el desarrollo y concepción de aplicaciones hipermedia. El desarrollo pasa por identificar las fuentes de datos y definir el documento de integración que da lugar al esquema federado. Sobre el esquema federado definimos las posibles *"navegaciones"* que recoge el documento de navegación, y finalmente, este documento sirve de referencia para obtener el documento de presentación. La especificación totalmente declarativa de los distintos aspectos de la aplicación, facilita tanto su desarrollo como su posterior mantenimiento. Incluir nuevos documentos o definir nuevos enlaces se limita a una rápida labor de edición. Además la utilización de XML permite reducir drásticamente las conexiones con el servidor, ya que toda la generación de páginas HTML que conforman *"el look"* de la aplicación, se genera dinámicamente en el cliente.

Entre los aspectos que acometeremos en un futuro se incluyen, la problemática de la heterogeneidad semántica entre documentos XML, la personalización de la integración y la presentación, y el soporte de arquitecturas *"push"*.

Referencias

[1] S. Abitebout. *On views and xml*. Sigmod Record, 28(4):30-38, 1999.

[2] J. Clark. *XSL Transformations (XSLT). Version 1.0*. http://www.w3.org/TR/xslt/, 1999.

[3] J. M. Blanco. *Integración de bases de datos relacionales por medio de un sistema terminológico: una propuesta utilizando BACK*. Tesis Doctoral, Universidad del País Vasco. Dpto. Lenguajes y Sistemas Informáticos, 1994.

[4] S Ceri, P. Fraternalli, and S. Paraboschi. *Data driven one-to-one web site generation for data-intensive applications*. Intl. Conf. on Very Large Data Bases (VLDB), 1999.

[5] S. DeRose, E. Maler, D. Orchard, and B. Trafford. *XML Linking Language (XLink).* http://www.w3.org/TR/xlink/, 1998.

[6] A. Deutsch, M. Fernandez, A. Levy, and D. Suciu. *XML-QL: A Query Language for XML.* http://www.w3.org/TR/NOTE-xml-ql/, 1998.

[7] J. Nielsen. *Hypertext and Hypermedia*. Academic Press, 1990.

[8] J. Robie, J. Lapp, and D. Schach. *XML Query Language (XQL).* http://wwww.w3.org/TandS/QL/QL98/pp/xql.html, 1998.

[9] G. Ross, D. Schwabe, and F. Lyardet. *Designing hypermedia applications with objects and patterns*. In International Journal of Software Engineering and Knowledge Engineering, volume 9, pages 745-766. World Scientific Publishing Company, 1999.

[10] P. Sheth and J. A. Larson. *Federated database systems for managing distributed heterogeneous and autonomous databases*. ACM Computing Surveys, 22(3):183-236, 1990.