UNIVERSIDAD DE SALAMANCA
FACULTAD DE TRADUCCIÓN Y DOCUMENTACIÓN
GRADO EN TRADUCCIÓN E INTERPRETACIÓN

Trabajo de Fin de Grado

Creación de filtros para documentos XML en la plataforma SDL Trados Studio 2009.

Guía práctica para traductores

Realizado por: Raquel Aranzana González

Dirigido por: Emilio Rodríguez Vázquez de Aldana

Salamanca, 2012

UNIVERSIDAD DE SALAMANCA

FACULTAD DE TRADUCCIÓN Y DOCUMENTACIÓN

GRADO EN TRADUCCIÓN E INTERPRETACIÓN

Trabajo de Fin de Grado

CREACIÓN DE FILTROS PARA DOCUMENTOS XML EN SDL TRADOS STUDIO 2009

Guía práctica para traductores

Vº Bº

Prof. Emilio Rodríguez Vázquez de Aldana

Salamanca, 2012

Resumen

En los últimos años, el mundo de la informática ha sido testigo de la expansión del lenguaje **XML**, utilizado para representar cualquier tipo de información. Por extensión, el traductor/localizador encuentra que, entre sus encargos, cada vez más documentos que traducir le son remitidos en este formato. En el presente trabajo, exponemos la forma de abordar la creación de filtros con la herramienta SDL Trados Studio 2009, que propone el lenguaje estándar **XPath** como mecanismo de selección de las partes que traducir de cualquier documento XML, independientemente de su estructura y organización del contenido.

Palabras clave: Traducción de documentos XML, XPath, Filtros para XML en SDL Trados Studio 2009.

Abstract

XML language, used to represent any type of information, has become widespread in the IT field in the last years. As a result, it is increasingly common for translators/localizers to receive assignments in XML format. The purpose of this essay is to show translators how to create filters in SDL Trados Studio 2009, a CAT tool that employs **XPath** language in order to select the translatable parts of a XML document without taking into account its structure or the way in which information is organized.

Key words: Translation of XML documents, XPath, SDL Trados Studio 2009 filters for XML documents

Índice

Índice de figuras

1. Introducción

En los últimos años, el florecimiento de Internet y de las nuevas tecnologías ha propiciado la proliferación de, entre otros, los documentos en formato XML (*eXtensible Markup Language*). Este proceso ha ido ligado a la necesidad cada vez mayor de localizar los productos informáticos en un mundo cada vez más globalizado.

Dicho auge ha dificultado enormemente la tarea de la localización, pues hasta hace poco los desarrolladores se veían obligados a crear un software interno que procesara el tipo concreto de archivo con el que estaban trabajando o a adaptar sus documentos a los programas de los que ya disponían, con el consiguiente derroche de tiempo y recursos.

Por este motivo, en 2002 el consorcio OASIS desarrolló un estándar de localización llamado XLIFF (OASIS, 2008), basado en la estructura de los archivos XML. Para este estándar, los documentos están formados por la conjunción de información localizable e información de estructura y lo que hace es separar estas dos partes para presentarle al traductor sólo aquello que debe traducir (Raya, 2004).

En la actualidad, la mayoría de herramientas TAO emplean XLIFF para abordar la traducción de documentos en diferentes formatos (Morado, Filip, 2012). De este modo, la función de los filtros en estos programas consiste en convertir los archivos de un formato cualquiera a XLIFF, para poder procesarlos y traducirlos.

Lo que estos hacen es, simplemente, transformar un documento en un formato cualquiera al estándar XLIFF, separando la parte localizable de los documentos de su estructura. Una vez que hemos traducido el contenido en nuestra herramienta TAO, el

programa vuelve a unir ambos tipos de información para generar un documento de destino exactamente igual que el original, pero en otro idioma.

En la actualidad, las herramientas TAO disponen de filtros estándar que sirven para procesar los tipos más habituales de archivos. Por ejemplo, SDL Trados Studio 2009 (SDL, 2009) dispone de un filtro para los ficheros en formato «docx» (Microsoft Word 2010), basado en el lenguaje XML, que separa el formato del texto traducible. Sin embargo, en otras ocasiones en las que el traductor dispone de documentos menos comunes, le corresponde a él crear sus propios filtros para que la herramienta TAO que esté empleando le muestre sólo las partes que desee traducir.

Dicho de otro modo, en muchos casos el cliente le entrega al traductor un documento XML para traducir con una estructura y organización particular. En tales ocasiones, es el propio traductor el que ha de buscar la manera de convertir su fichero al formato XLIFF de modo que pueda ser traducido en una herramienta TAO cualquiera, y esto se hace mediante la creación de filtros.

Por este motivo, el propósito de este Trabajo de Fin de Grado es proporcionar unas directrices básicas que ayuden al traductor a comprender el proceso de creación de filtros para documentos XML en SDL Trados Studio 2009. Una vez entendidas y asimiladas, podrá aplicar sus conocimientos a cualquier archivo XML que tenga que traducir en el ejercicio de su profesión. Así, nuestra intención es que este texto sirva de guía introductoria tanto a los profesionales de la traducción como, en un ámbito más cercano y restringido, a los futuros estudiantes de la asignatura optativa de «Localización (Inglés)» del Grado de Traducción e Interpretación de esta Facultad.

Por tanto, lo que buscamos conseguir con este trabajo es exponer nuestra labor de experimentación con un software y un lenguaje informático determinados con los que realizamos un conjunto de prácticas en dicha asignatura. Por otra parte, también deseamos dar a conocer a los traductores este extraño mundo de los lenguajes formales de ordenadores de una forma comprensible para dicho entorno.

Para este fin, hemos dividido este trabajo en tres apartados complementarios entre sí. En primer lugar, proporcionaremos una definición del lenguaje XML y de las características de los documentos en este formato. A continuación, explicaremos en qué consiste el lenguaje XPath y cuál es su utilidad de cara a procesar documentos XML y a crear filtros para herramientas TAO.

Finalmente, profundizaremos en el proceso de la creación de los filtros en SDL Trados Studio 2009. Para ello, nos serviremos de ejemplos de diversa índole que ilustrarán la mayoría de necesidades que le pueden surgir a un traductor a la hora de enfrentarse a un documento XML.

2. El lenguaje XML

2.1. Origen

El lenguaje XML (*Extensible Markup Language* cuya traducción adaptada al español es «Lenguaje Extensible de Marcas») es un estándar internacional desarrollado por el Comité de Revisión Editorial de SGML (*Standard Generalized Markup Language*) en el año 1996. Dicho comité llevó a cabo este proyecto bajo la supervisión del *World Wide Web Consortium* (W3C, en adelante), un consorcio internacional que emite recomendaciones para la red informática mundial.

El Comité de Revisión Editorial de SGML ya había desarrollado en los años 80 un lenguaje llamado, precisamente, SGML, cuyo propósito era separar en un documento el contenido del formato. Sin embargo, este lenguaje era demasiado complejo y, en 1992, fue sustituido por el HTML (*HyperText Markup Language)*, que era más fácil de aplicar, pero menos completo. En 1996, el grupo decidió crear un lenguaje que combinara la riqueza del SGML con la facilidad de aplicación del lenguaje HTML, y el resultado de esta unión fue el lenguaje XML (W3C, 2012a).

2.2. Los lenguajes de etiquetas HTML y XML: similitudes y diferencias

El trabajo desarrollado por el Comité aparece explicado en la denominada «Recomendación de XML» (W3C, 2008). En este texto se define al lenguaje XML del siguiente modo: «El Lenguaje Extensible de Marcas, abreviado XML, describe una clase de objetos de datos llamados documentos XML y parcialmente describe el comportamiento de programas de computador que pueden procesarlos». (W3C, 2008)

Como veremos a continuación, el lenguaje XML no es más que una propuesta sintáctica sobre cómo estructurar la información de los documentos. Es decir, su finalidad es ofrecer una forma estándar de organizar el contenido de los documentos para que, posteriormente, puedan ser procesados con un mismo conjunto de técnicas. Para cumplir su cometido, a saber, estructurar un texto o, mejor, un conjunto de cualquier tipo de información, XML utiliza el mismo sistema de marcas que el lenguaje HTML. Como ya sabemos, éstas, a las que también se puede llamar etiquetas, aparecen delimitadas por los símbolos «<» y «>». Mostramos a continuación un pequeño ejemplo de un documento HTML y otro XML para observar su apariencia similar.

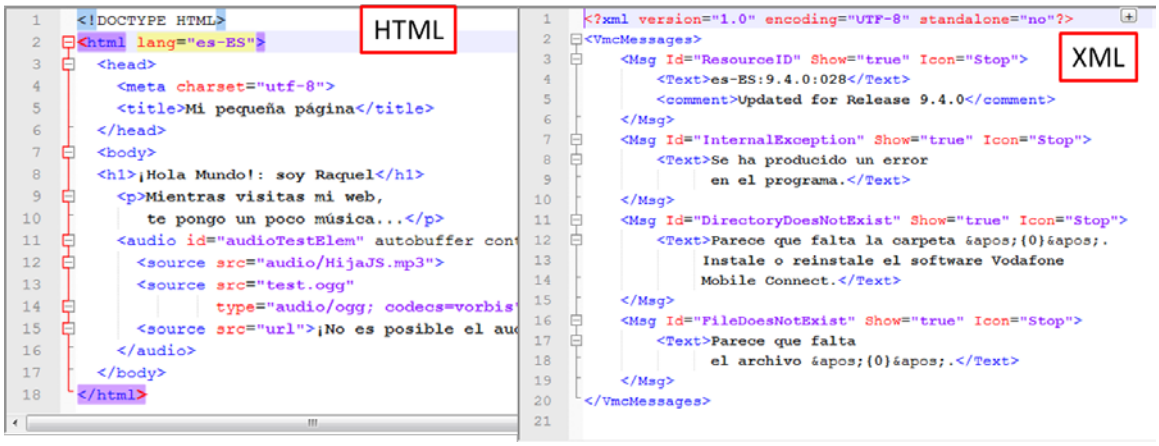

Figura 2.1: Similitudes entre un fichero HTML y XML.

Como podemos observar en la Figura 2.1, los documentos HTML y XML organizan los textos (lo que en la ilustración aparece resaltado en color negro) con un sistema de etiquetas, que aparecen en la imagen en color azul. Si observamos más atentamente la figura, vemos que algunas etiquetas contienen antes de su símbolo de finalización («>»), una o más palabras en color rojo.

Estas marcas rojas se denominan en ambos lenguajes «atributos». A ellos les sigue el símbolo de igualdad («=») y, entre comillas, a su vez, una tira de caracteres en

color morado, que es el valor que se le asocia a ese atributo concreto. Es el caso, en la figura anterior, de las etiquetas «meta», «source» e «Id», entre otras. Por tanto, como vemos, una etiqueta puede contener una serie de atributos con distintos valores tanto en el lenguaje HTML como en el XML.

Para finalizar el análisis de las semejanzas entre ambos lenguajes, podemos observar en dicha figura que a algunas etiquetas las precede, después del símbolo «<», el carácter «/». Tal es el caso, por ejemplo, de las etiquetas «</html>», «</head>» y «</Msg>».

Éstas, llamadas etiquetas de cierre, marcan el final de un bloque del documento cuyo inicio será la etiqueta que, con el mismo nombre y sin el símbolo «/», la precederá en el mismo. Así, en la Figura 2.1, entre otros bloques, podemos ver la estructuras <html>...</html>, <head>...</head> o <Msg>…</Msg>.

La principal diferencia entre ambos lenguajes radica en su finalidad. El lenguaje HTML fue ideado para presentar información en un navegador web y XML, por el contrario, es un lenguaje –en realidad, un metalenguaje- propuesto para estructurar información de cualquier tipo.

Por otro lado, en el lenguaje HTML, el conjunto de etiquetas es prexistente. Esto implica que han sido previamente definidas por el organismo normalizador (W3C, 2012b) y que los navegadores, que incorporan un intérprete del lenguaje HTML, transforman el significado de las mismas en una presentación determinada del documento escrito en dicho lenguaje. Para aclarar lo que venimos diciendo, obsérvese la Figura 2.2.

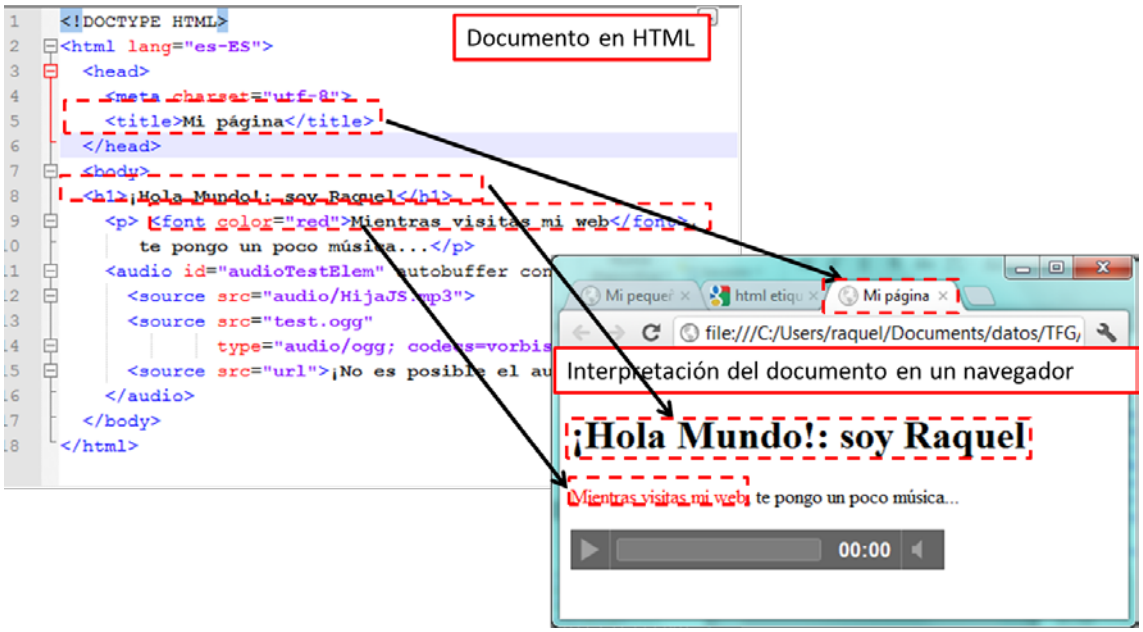

Figura 2.2: Documento en lenguaje HTML e interpretación en un navegador.

En el documento HTML de la figura de arriba hemos resaltado tres etiquetas con el texto que incorporan cada una («title», «h1» y «font», respectivamente) y cómo éstas se interpretan en un navegador. Por ejemplo, el intérprete HTML de los navegadores determina que el texto de la etiqueta <title> se sitúe en la pestaña del navegador correspondiente a la página que se está mostrando. Los navegadores, por seguir con el ejemplo, cuando encuentran una etiqueta <h1> determinan que el texto que contengan se presente resaltado con un tamaño de letra más grande que el resto.

Para finalizar, hemos marcado también una etiqueta <font>, a la que hemos asociado un atributo «color» con valor «red». Como puede verse, el navegador lo interpreta poniendo el conjunto del bloque de palabras en dicho color (rojo). En resumidas cuentas, como hemos indicado anteriormente, el significado de las etiquetas en HTML, que es conocido por los navegadores, tiene un efecto visual de presentación de los documentos en los navegadores.

Por el contrario, en XML las etiquetas no están predefinidas. En realidad, las idea el creador del documento para estructurarlo con vistas a procesar posteriormente y de manera automática la información en él contenida. En la Figura 2.3 mostramos un nuevo documento XML, distinto del presentado en la Figura 2.1. Como puede observarse, no comparten ninguna etiqueta. Los creadores de cada documento han elaborado unas para organizar la información que contienen.

Centrémonos en el documento de la Figura 2.3: podemos colegir, con lo que mostramos, que la persona que ha creado este documento ha decidido organizar en una etiqueta «segment» un texto en inglés con su correspondiente traducción al español. Para delimitar con etiquetas qué texto está en inglés y cuál en español, el diseñador ha asociado a la etiqueta «source» (que como vemos agrupa los textos de ambos idiomas), un atributo «lang» que toma el valor «enGB» o «esES» según cuál sea el idioma del texto.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <document>
3      <segment id="277">
4          <modified date="2011-12-31" hour="18:52:15">EULA</modified>
5          <text approved="yes">
6              <source lang="enGB">
7                  Translation Teams for Joomla!1.5 may have also releas
8                  installation packages where site, administrator and s
9              </source>
10             <source lang="esES">
11                 Translation Teams for Joomla! 1.5 may have also relea
12                 installation packages where site, administrator and s
13             </source>
14         </text>
15     </segment>
```

Figura 2.3: Fragmento de un documento XML.

Los lenguajes XML y HTML no sólo se diferencian en su función sino también en ciertos aspectos sintácticos. XML es un lenguaje estrictamente jerárquico, lo cual

implica que las etiquetas no se pueden solapar. Dicho de otro modo, en XML todas las etiquetas deben cerrarse en el mismo orden en el que se abrieron.

Como puede verse en la Figura 2.4, en la primera estructura las etiquetas <X> e <Y> se encuentran solapadas, lo que no es una construcción válida en un documento XML bien formado. En la construcción inferior de la misma figura observamos la estructura correcta para ese par de etiquetas: la etiqueta «Y», hija de «X» ha de cerrarse antes de que se cierre cualquier etiqueta ascendiente suya.

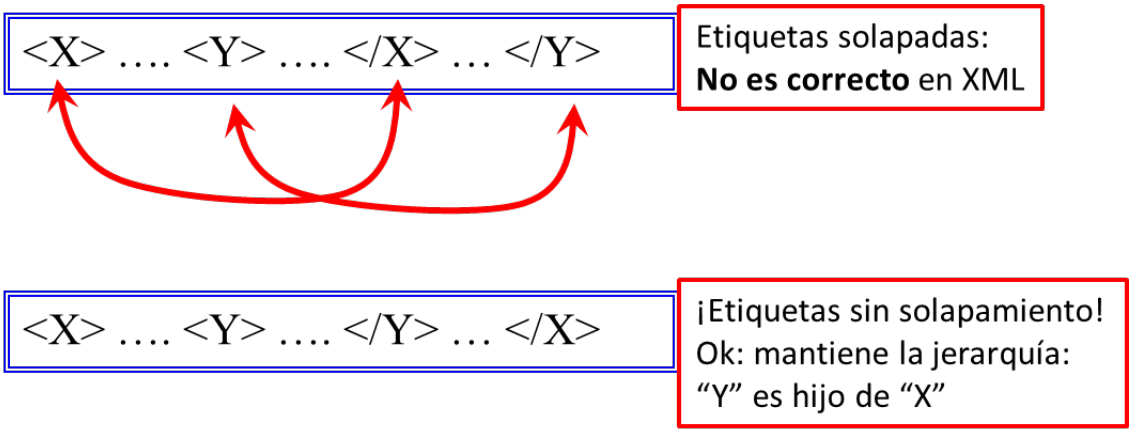

Figura 2.4: Ejemplo de etiquetas solapadas e incorrectas en XML y etiquetas correctamente conformadas.

Si esta regla se incumple, la aplicación con la que se desee procesar el documento no será capaz de hacerlo y emitirá un error de mala formación del mismo. Por ejemplo, en la Figura 2.5 mostramos un documento en el que se abre una etiqueta <asignatura> que carece de la correspondiente etiqueta cierre (</asignatura>). Dado que el lenguaje XML es, como se acaba de mencionar, un lenguaje estrictamente jerárquico, al abrir este fichero, por ejemplo, con un navegador web, éste en vez de presentarlo mostrará un mensaje de error indicando su origen (ver Figura 2.6).

Figura 2.5: Documento XML mal conformado.

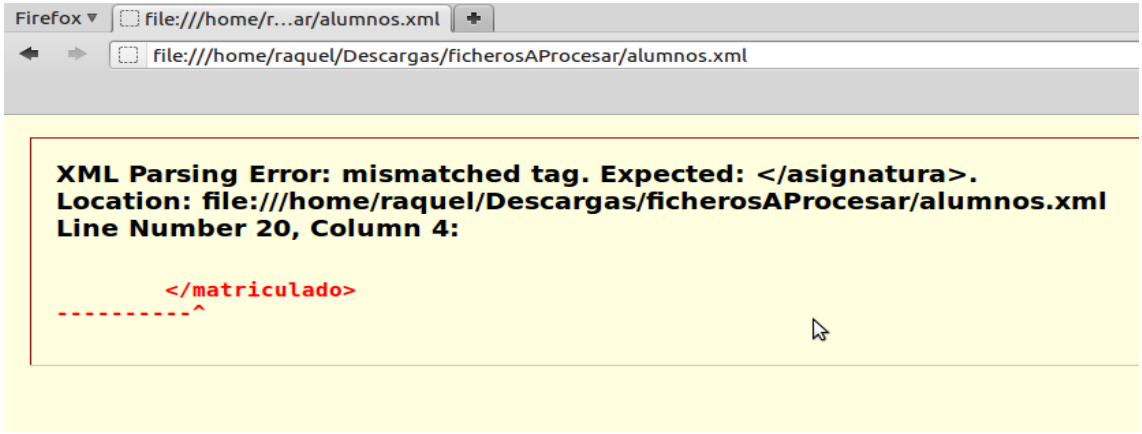

Figura 2.6: Resultado del análisis de un documento XML mal formado.

Sin embargo, si todas las etiquetas se abren y cierran siguiendo un orden jerárquico, el navegador mostrará el documento XML tal y como aparece en el editor. En la Figura 2.7 mostramos el documento anterior debidamente conformado (se ha cerrado la «asignatura» en el lugar adecuado) y la salida que devuelve el navegador en la Figura 2.8**.**

```
<?xml version="1.0" encoding="iso-8859-1"?>
<alumnos>
<alumno>
        <quien>
                <apellidos>Rodríguez</apellidos>
                <nombre>Juana</nombre>
        </quien>
        <matriculado>
        <asignatura código="1212">
                <nombre>Localización (Inglés)</nombre>
                <créditos>6</créditos>
</asignatura>
        <asignatura código ="1214">
                <nombre>Terminología</nombre>
                <créditos>6</créditos>
        </asignatura>
        <asignatura código="1213">
                <nombre>Recursos Tecnológicos</nombre>
                <créditos>6</créditos>
        </asignatura>
        </matriculado>
```

Figura 2.7: Documento XML sintácticamente correcto.

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<alumnos>
 -<alumno>
  -<quien>
    <apellidos>Rodríguez</apellidos>
    <nombre>Juana</nombre>
   </quien>
  -<matriculado>
   -<asignatura código="1212">
     <nombre>Localización (Inglés)</nombre>
     <créditos>6</créditos>
    </asignatura>
   -<asignatura código="1214">
     <nombre>Terminología</nombre>
     <créditos>6</créditos>
    </asignatura>
   -<asignatura código="1213">
     <nombre>Recursos Tecnológicos</nombre>
     <créditos>6</créditos>
    </asignatura>
   </matriculado>
  -<dirección>
   -<dir_curso>
    -<calle>
```

Figura 2.8: Vista de un documento XML correctamente formado en un navegador.

De estos ejemplos que hemos mostrado nos interesa destacar también dos aspectos. Por un lado, que los navegadores modernos incorporan un analizador sintáctico que examina la correcta estructuración de cualquier documento XML. Es

decir, los podemos utilizar como herramienta para validar si el documento que estamos editando está adecuadamente construido.

Por otro, que un navegador, al recibir un documento XML correctamente elaborado, se limita a mostrar en la pantalla tanto los datos propiamente dichos del mismo como las etiquetas que los estructuran. Como hemos mostrado anteriormente, esto no ocurre en un documento HTML: las etiquetas de éstos se «interpretan» en el navegador produciendo un efecto visual y/o de organización del documento en la ventana correspondiente.

Siguiendo con las diferencias entre XML y HTML a nivel sintáctico, simplemente nos queda comentar que el lenguaje HTML es mucho más flexible y permisivo que el XML. De esta manera, si a una etiqueta abierta no le prosigue su cierre, el navegador, al interpretar dicho documento HTML, tomará la determinación de finalizarlo y presentará dicho documento en la ventana abierta. Así, en la Figura 2.9 puede observarse el código fuente de una página web con una etiqueta <tr> sin cierre y cómo el navegador, pese a ello, interpreta el código y muestra la página web.

Figura 2.9: Código fuente HTML de un documento con una etiqueta sin cierre y, pese a ello, su interpretación en un navegador.

2.3. Partes de un documento XML

Ahora que hemos presentado el lenguaje XML, en este apartado vamos a explicar con más detalle las partes básicas de un documento XML. No obstante, no pretendemos proporcionar una descripción exhaustiva, sino sólo una breve introducción. Para una mayor profundización se puede consultar la amplia bibliografía que hay al respecto. Nosotros, como medio de documentación para abordar este capítulo, hemos consultado, en concreto, (Lecomte, Boulanger, 2009) y (Gutiérrez, 2001).

Como hemos podido observar, el primer componente lo constituyen las etiquetas mediante las que se organiza y estructura la información, mientras que el segundo es, simplemente, la «verdadera información» del documento. En otras palabras, un

documento XML está constituido, por un lado, por las etiquetas y, por otro, por los datos que aparecen entre éstas. Estas marcas o etiquetas están estructuradas en árbol, lo que facilita determinar la jerarquía de la información que en él se presenta.

Para la descripción de las partes nos vamos a valer del ejemplo que aparece en la siguiente figura:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <document>
3      <message id="277">
4          <date>2011-12-31 18:52:15</date>
5          <modified_by>EULA</modified_by>
6          <text lang="en">
7              <weblink address="http://www.joomla.org"
8                       logo="http://www.joomla.org/images/joomla.jpg"
9                       alt="Logo of Joomla!">Joomla!</weblink> is a
10             <emphasis translate="yes">free open source</emphasis> framework
11             and content publishing system designed for quickly creating highly interactive
12             <emphasis translate="yes">multi-language Web</emphasis> sites, online communities,
13             media portals, blogs and eCommerce applications.
14         </text>
15         <!-- Text in Spanish -->
16         <text lang="es">
17             <weblink address="http://www.joomla.org"
18                      logo="http://www.joomla.org/images/joomla.jpg"
19                      alt="Logo of Joomla!">Joomla!</weblink> is a
20             <emphasis translate="yes">free open source</emphasis> framework
21             and content publishing system designed for quickly creating highly interactive
22             <emphasis translate="yes">multi-language Web</emphasis> sites, online communities,
23             media portals, blogs and eCommerce applications.
24         </text>
```

Figura 2.10: Partes de un documento XML

Prólogo

Éste es el nombre que recibe la primera línea de todo documento XML y que, en la Figura 2.10, es en concreto la línea 1. En el **prólogo** se declara que el archivo es un documento XML («*<?xml*») y se detalla la versión del lenguaje XML utilizada para la composición del documento (en este caso, «*versión=”1.0”*»). Como podemos observar, también queda expresado en este prólogo la codificación de caracteres empleada en el documento (en nuestro ejemplo, «*encoding=”UTF-8”*»).

Elementos

Los **elementos** son los que forman el esqueleto en árbol de un documento XML. Estos aparecen delimitados por una etiqueta de apertura y una de cierre (<*message*> </*message*> en el caso del documento de la Figura 2.10, por ejemplo) entre las que se incluye la información deseada.

Al elemento principal en la jerarquía en árbol de un documento XML se le denomina **elemento raíz**. En el caso del documento mostrado en la Figura 2.10, el elemento raíz es <*document*>.

Hay que tener un cuidado especial a la hora de asignar un nombre a los distintos elementos. En primer lugar, es preciso tener en mente que el lenguaje XML hace distinción entre mayúsculas y minúsculas (es *case sensitive*), por lo que hay que emplearlas de una manera homogénea a lo largo de todo el documento. Por otro lado, en el nombre de los elementos no se pueden incluir espacios. De este modo, sí que sería válido, por ejemplo, el nombre «<documentname>», pero no «<document name>».

Atributos

Los **atributos** presentan información adicional que se asocia a los elementos para, por ejemplo, definir el idioma en que se presenta la información o la fecha en que se publicó un libro. No tienen por qué aparecer en todos los documentos XML y, en caso de que se introduzcan, tampoco es necesario asociarlos a todos y cada uno de los elementos. Así, en el documento XML presentado en la Figura 2.10, son atributos con un valor asociado, por ejemplo, «*lang=”enGB”*» y «*lang=”esES”*».

Comentarios

Siempre que quieran, los creadores del documento pueden añadir **comentarios** en cualquier parte del mismo. Para ello, como puede observarse en la línea 15 del documento presentado en la Figura 2.10, el texto del comentario debe ir insertado entre la combinación de caracteres «<*!*--» y «-->». Las aplicaciones no procesan estos comentarios, pero pueden leerse al editar el documento con los programas empleados para ello.

2.4. Normas sintácticas para la construcción correcta de un documento XML

Como comentamos en el apartado 2.1 de este Trabajo, el lenguaje XML es mucho más riguroso, a nivel sintáctico, que el lenguaje HTML. Para una correcta composición de un documento acorde con las normas de este lenguaje, se deben de cumplir los siguientes requisitos:

- El prólogo debe ser válido.
- Siempre tiene que haber un elemento raíz que englobe todo el documento.
- Todas las etiquetas deben estar cerradas.
- Las etiquetas deben de mantener la jerarquía (es decir, no pueden, como dijimos en el apartado 1.2, solaparse).
- Los valores de los atributos deben estar entre comillas.
- Los nombres de los elementos no pueden contener caracteres no permitidos (como, por ejemplo, espacios).

3. El lenguaje de selección para XML: XPath.

3.1. Introducción

XPath (abreviación de «XML Path Language») es un lenguaje que permite, mediante expresiones sintácticas, seleccionar subconjuntos concretos de un documento XML (W3C, 2011). Dicho de otro modo, lo que hace este lenguaje es navegar por la estructura en árbol de los documentos XML hasta llegar a las partes que se le indican en las expresiones.

XPath es lo que suelen emplear las herramientas TAO para crear filtros para documentos XML, por lo que resulta de suma importancia para los traductores conocerlo y saber utilizarlo. Además, se trata precisamente del lenguaje del que se sirve SDL Trados Studio 2009 para la creación de filtros, por lo que antes de empezar con ellos es necesario explicar su funcionamiento.

Dado el propósito eminentemente práctico de este trabajo, a saber, proporcionar a los traductores los conocimientos precisos para elaborar filtros para documentos XML, no hemos considerado oportuno perdernos en densas descripciones de la sintaxis del lenguaje XPath. En su lugar, hemos preferido explicar los fundamentos de este lenguaje mediante el llamado método inductivo: partiendo de ejemplos concretos para llegar a la comprensión de las distintas normas. Con ello, los traductores podrán adquirir la base necesaria para poder enfrentarse a cualquier documento XML, sea cual sea su extensión o complejidad.

3.2. Procesador XPath seleccionado

De cara a ilustrar los ejemplos elegidos para este apartado, nos hemos servido del editor de documentos XML **EditiX**[1]. Se trata de un programa que tiene el lenguaje XPath incorporado y que, mediante éste, procesa los ficheros XML y permite, entre otras cosas, modificarlos. Aconsejamos al traductor que se sirva de esta herramienta cuando vaya a crear filtros para cualquier programa de traducción asistida, pues le permite comprobar la validez de las expresiones XPath que vaya a emplear antes de introducirlas en su herramienta TAO.

Por supuesto, EditiX no es más que una opción entre todo el software disponible. Otra herramienta del mismo tipo que también recomendamos es **XPath Expression Testbed**, programa en línea disponible en http://www.whitebeam.org/library/guide/TechNotes/xpathtestbed.rhtm. Esta última tiene la ventaja de que no requiere la instalación de ningún software en el ordenador pero, como contrapartida, ofrece menos prestaciones a sus usuarios y tiene una interfaz menos agradable.

3.3. XPath a través de ejemplos prácticos

Como ya hemos explicado anteriormente, los archivos XML disponen su información en una estructura en forma de árbol. En este apartado, nos serviremos de esquemas arbóreos para presentar el esqueleto básico de algunos de los documentos XML con los que vamos a trabajar similares al que se presenta en la Figura 3.1.

[1] Es posible descargar una versión gratuita de EditiX de la página web http://free.editix.com/. En el Anexo I de este Trabajo se incluye un breve manual de uso para orientar al traductor en el manejo de esta herramienta.

Así, imaginemos que disponemos de un archivo cuya estructura se corresponde a la mostrada en el esquema de la Figura 3.1. En primer lugar, conviene aclarar que el asterisco presente junto a los nodos «alumno» y «asignatura» quiere decir que su número es variable, aunque en el esquema sólo aparezcan una vez.

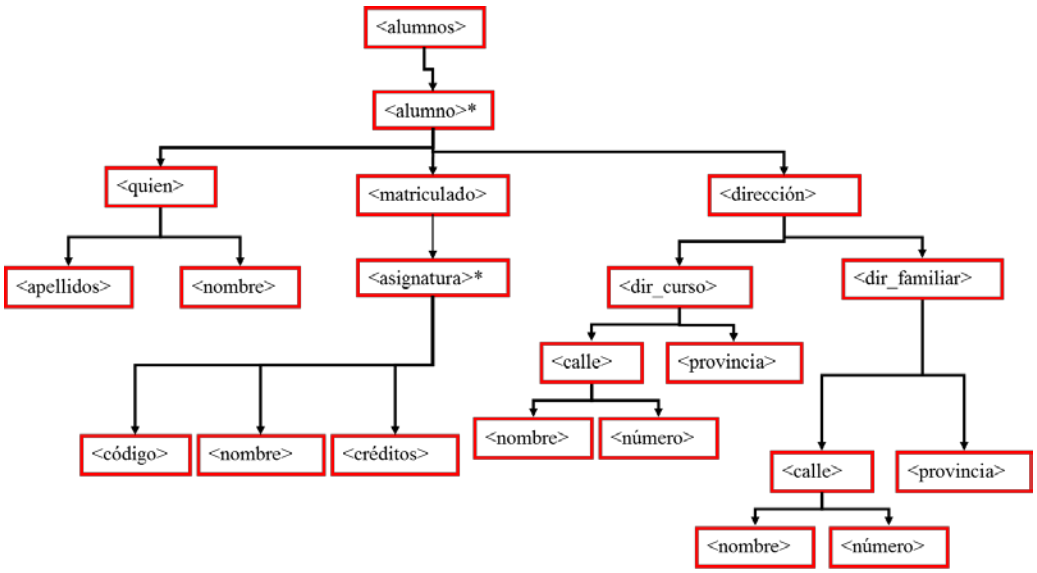

Figura 3.1: Árbol de un documento XML.

Una vez dicho esto, y antes de intentar entender la lógica del lenguaje XPath, es preciso volver a lo ya explicado para entender el modo en que este archivo está organizado. En la Figura 3.1 podemos ver que el nodo raíz del documento que nos atañe ahora es «alumnos», del cual parten otros nodos llamados «alumno». De cada uno de estos nodos «alumno» nacen, a su vez, tres nodos hijos: «quien», «matriculado» y «dirección».

Para continuar, el nodo «quien» tiene dos hijos («apellidos» y «nombre»); el nodo «matriculado», uno o varios, llamados siempre «asignatura», cuyo número dependerá, se supone, de la cantidad de materias en las que esté matriculado el alumno; y, por último, de «dirección» parten dos hijos («dir_curso» y «dir_familiar»). Cada uno

de estos nodos tiene otros nodos hijos, pero, ahora que se ha dejado clara la lógica del documento, consideramos que no es preciso prolongar la explicación.

Supongamos que, para comenzar, deseamos seleccionar todos los nodos «alumno», hijos del nodo raíz «alumnos», del documento que, como ya hemos explicado, puede contener un número indefinido de ellos. Para ello, deberíamos escribir la expresión XPath **/alumnos/alumno**. En ella, la primera barra indica que la búsqueda empieza desde el nodo raíz del documento (en este caso «alumnos») para, a continuación, recuperar todos los nodos «alumno» descendientes suyos.

Imaginemos que deseamos ser más precisos en nuestra extracción y que, en lugar de querer seleccionar todo el nodo «alumno», nuestra intención es hacernos sólo con el nodo «quien», hijo de «alumno» y nieto de «alumnos». Para ello, y siguiendo la lógica del ejemplo anterior, la expresión XPath que introduzcamos debería ser **/alumnos/alumno/quien**.

Para continuar, en el caso de que dentro del nodo «quien» queramos seleccionar únicamente el nodo hijo «apellidos», tenemos dos opciones. La primera consiste en elaborar nuestra expresión XPath de acuerdo con la lógica mostrada en los ejemplos anteriores, con lo que escribiríamos **/alumnos/alumno/quien/apellidos**.

Sin embargo, el lenguaje XPath dispone de la expresión «//», que sirve para seleccionar todos los nodos llamados de una determinada forma, sin importar el lugar del árbol del documento en el que se encuentren. De este modo, podríamos emplear la expresión **//apellidos,** más sencilla, para seleccionar los nodos «apellidos» de nuestro documento.

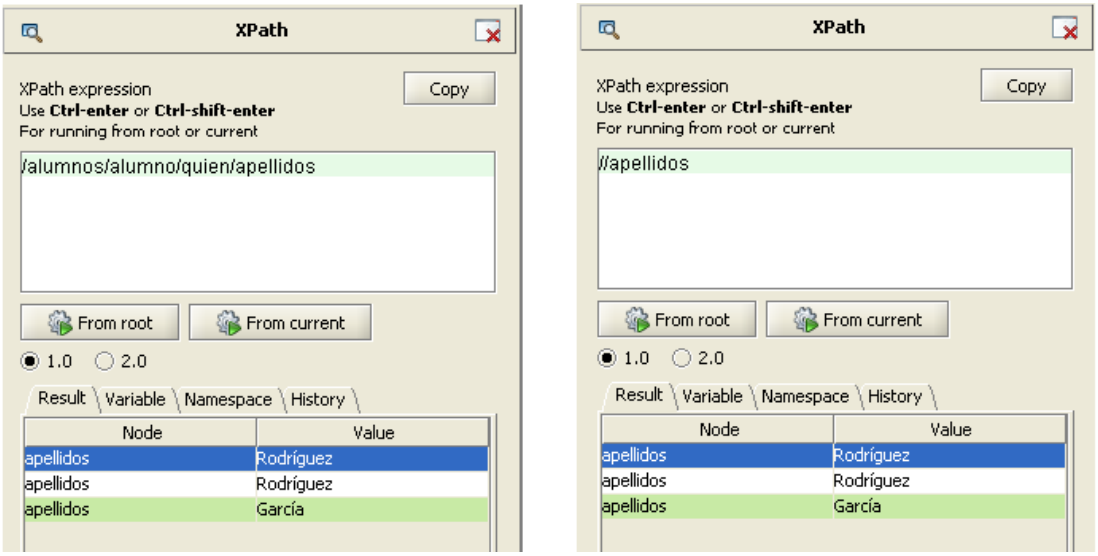

Figura 3.2: Mismos resultados en EditiX de dos expresiones XPath distintas.

Esto implica que, para llegar al nodo «apellidos» podemos servirnos tanto de la expresión que marca la ruta completa desde el nodo raíz hasta el que nos interesa, como de las dos barras, que nos llevarían directamente a «apellidos». Ambas fórmulas son válidas pero, como frente a cualquier problema, es preferible emplear la opción más sencilla. Por esta razón, en el caso que nos concierne lo ideal sería emplear la expresión **//apellidos**.

¿Y qué ocurre si en lugar del nodo «apellidos» lo que queremos es seleccionar todos los nodos «nombre» descendientes de «alumno»? Como se puede ver en la Figura 3.1, el documento con el que estamos trabajando contiene nodos «nombre» en distintas partes de su estructura en árbol. Así, hay nodos «nombre» descendientes de «quien», nodos «nombre» descendientes de «asignatura» y nodos «nombre» descendientes de «calle». Teniendo esto en cuenta, debemos recordar que si empleáramos la expresión **//nombre,** le estaríamos indicando a XPath que nos mostrara todos los nodos «nombre» presentes en el documento (ver Figura 3.3).

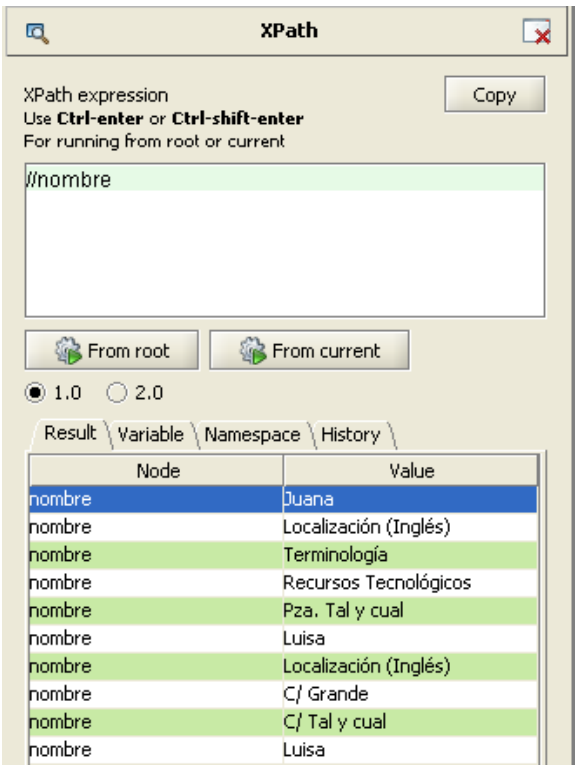

Figura 3.3: Resultados en EditiX de la expresión XPath //nombre.

Por esta razón, para este caso concreto la expresión que deberíamos emplear siguiendo los razonamientos anteriores es **/alumnos/alumno/quien/nombre.** Otra opción más breve consiste en utilizar **//quien/nombre**, con la seleccionaríamos todos los nodos «quien» (que, al contrario que «nombre», sólo son hijos de «alumno») y, dentro de ellos, el nodo «nombre».

Siguiendo con el mismo archivo, supongamos que ahora queremos seleccionar todos los nodos del documento hijos de «asignatura», a saber, «código», «nombre» y «créditos» (ver Figura 3.1). Como ya hemos explicado con anterioridad, el número de nodos «alumno» del documento es indeterminado, al igual que el número de nodos «asignatura».

Si tenemos en cuenta la lógica que hemos aprendido con los ejemplos anteriores, para seleccionar todos los descendientes de todos los nodos «asignatura» del

documento, deberíamos servirnos de tres expresiones. Éstas serían, en concreto: **//asignatura/código**, **//asignatura/nombre** y **//asignatura/créditos**. En este caso concreto, se da la circunstancia de que «asignatura» sólo tiene tres descendientes, pero ¿qué pasaría sin en vez de tres fueran treinta? Escribir una expresión por cada uno de ellos se convertiría en un trabajo pesado y engorroso.

De esta manera, nos bastaría la expresión **//asignatura/*** para seleccionar todos los nodos hijos de «asignatura», independientemente de su cantidad o denominación. Con ella, le estamos indicando a XPath que recupere todos los nodos «asignatura» del documento (**//asignatura**) y que, una vez hecho esto, muestre los nodos de todos sus descendientes (**/***), independientemente de cuántos sean o de su nombre.

Todos los ejemplos que hemos presentado hasta ahora tenían como fin seleccionar uno o varios nodos de un documento XML. No obstante, como ya se ha explicado anteriormente, los documentos XML también pueden contener atributos. Tal es el caso del archivo representado en la Figura 3.4, en el que observamos que todo el texto del documento se recoge en atributos.

```xml
<?xml version="1.0" encoding="Windows-1252" ?>
<NotepadPlus>
    <Native-Langue name="English" filename="english.xml" >
        <Menu>
            <Main>
                <!-- Main Menu Entries -->
                <Entries>
                    <Item id="0" name="File"/>
                    <Item id="1" name="Edit"/>
                </Entries>

                <!-- Sub Menu Entries -->
                <SubEntries>
                    <Item posX="1" posY="9"  name="Copy to Clipboard"/>
                    <Item posX="1" posY="10" name="Indent"/>
                </SubEntries>
            </Main>
        </Menu>
        <Dialog>
            <Find title="" titleFind="Find" titleReplace="Replace" titleFindInFiles="Find in Files" titleMark="Mark">
                <Item id="1"    name="Find Next"/>
                <Item id="2"    name="Close"/>
            </Find>
            <GoToLine title="Go to...">
                <Item id="2007" name="Line"/>
                <Item id="2008" name="Offset"/>
            </GoToLine>

            <Run title="Run...">
                <Item id="1903" name="The Program to Run"/>
            </Run>
            <Window title="Windows">
                <Item id="1" name="Activate"/>
                <Item id="2" name="OK"/>
            </Window>
        </Dialog>
    </Native-Langue>
</NotepadPlus>
```

Figura 3.4: Documento XML con texto en los atributos.

Imaginemos que queremos seleccionar todos los atributos «name» asociados a «Item» del documento, independientemente de qué nodo sea el padre de «Item». Como se ha visto en los ejemplos anteriores, la expresión con la que se puede obtener todos los elementos «Item», sin tener en cuenta en qué parte del documento se encuentran, es **//Item**. A ésta habrá que añadirle la mención al atributo, que en XPath se realiza mediante el símbolo de arroba. De este modo, habrá que prolongar expresión **//Item** a **//Item/@name** (ver Figura 3.5).

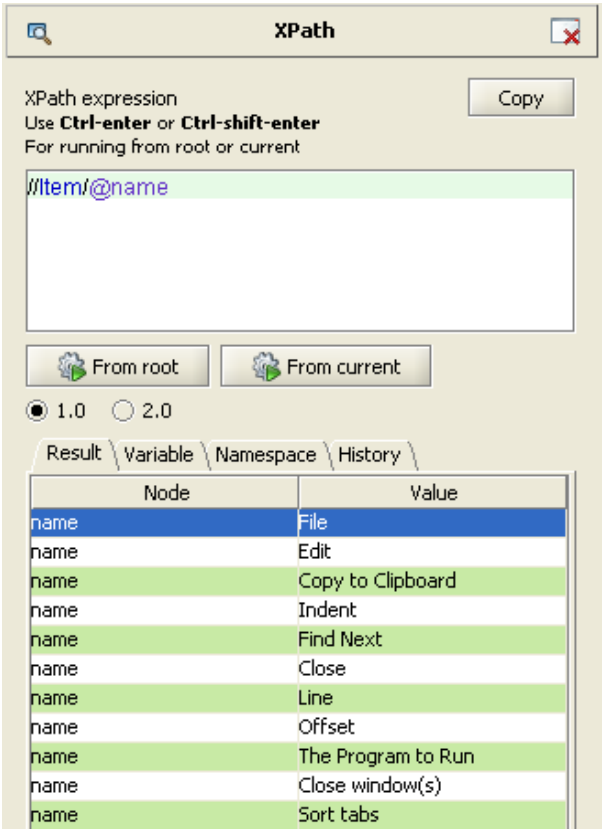

Figura 3.5: Resultados en EditiX de la expresión XPath //Item/@name.

Por otro lado, imaginemos que queremos afinar más nuestra selección y sólo deseamos obtener los atributos «name» de los nodos «Item» que sean descendientes del nodo «Entries». Para ello, sirviéndonos del razonamiento que hemos aprendido con otros ejemplos anteriores, la expresión que deberíamos emplear sería **//Entries/Item/@name** (ver Figura 3.6).

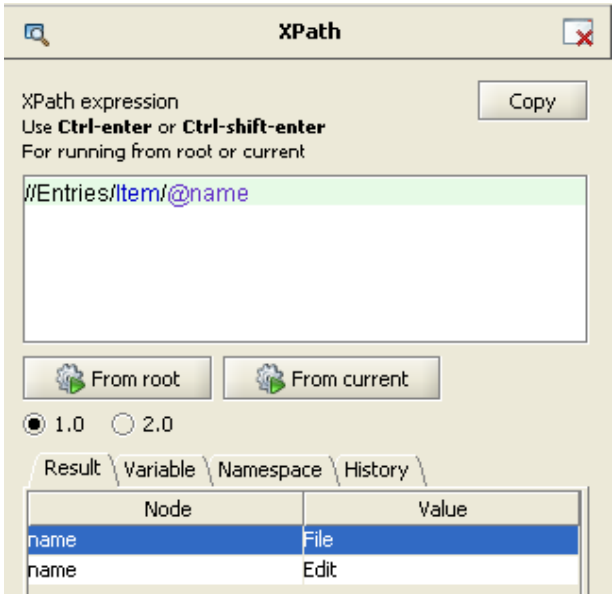

Figura 3.6: Resultados en Editix de la expresión XPath //Entries/Item/@name.

Ahora, supongamos que deseamos seleccionar todos los atributos «title» del documento que, como se puede ver en la Figura 3.4, se encuentran relacionados a los nodos «Find», «GoToLine», «Run» y «Window». Puesto que queremos recuperarlos todos, independientemente del nodo al que estén asociados, la expresión de la que nos debemos servir es **//@title** (ver Figura 3.7).

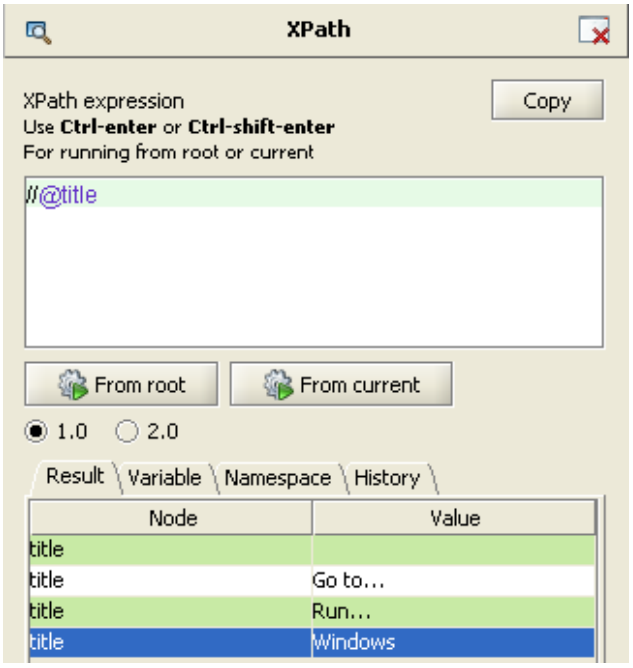

Figura 3.7: Resultados en EditiX de la expresión XPath //@title.

Otra situación que podría darse es que quisiéramos seleccionar todos los atributos asociados a un nodo determinado, independientemente de su nombre o cantidad. Supongamos que, en el caso del documento ilustrado en la Figura 3.4, nos interesara recuperar la totalidad de atributos asociados al nodo «Find». Con este objetivo, deberíamos emplear la expresión **//Find/@*** (ver Figura 3.8), en la que **//Find** indica a Xpath que su búsqueda debe partir de todos los nodos así llamados. A continuación, **/@*** le señala que debe recuperar todos los atributos relacionados con este nodo independientemente de su nombre o número.

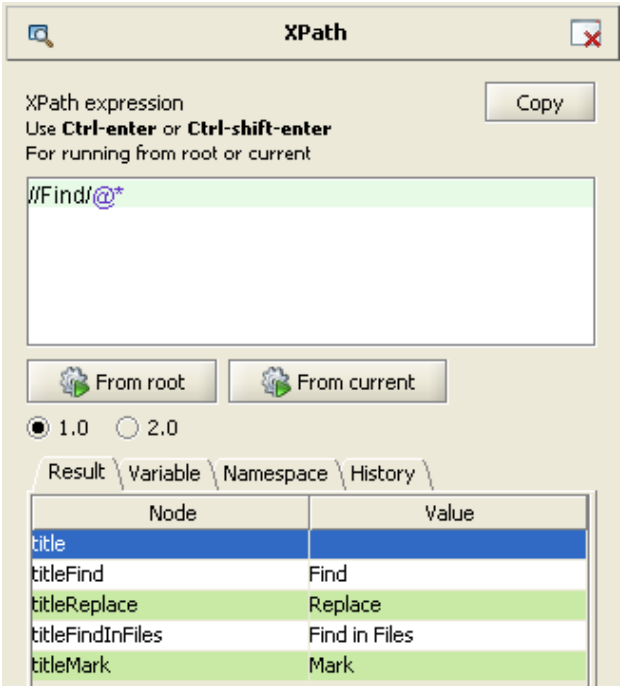

Figura 3.8: Resultados en EditiX de la expresión XPath //Find/@*.

Ahora que ya hemos entendido el razonamiento básico del lenguaje XPath es el momento de empezar a manejar documentos un poco más complejos. Supongamos que un cliente nos ha proporcionado el documento XML cuya estructura en árbol se muestra en la Figura 3.9, correspondiente a una página web (ver Figura 3.10), para que se lo traduzcamos al español.

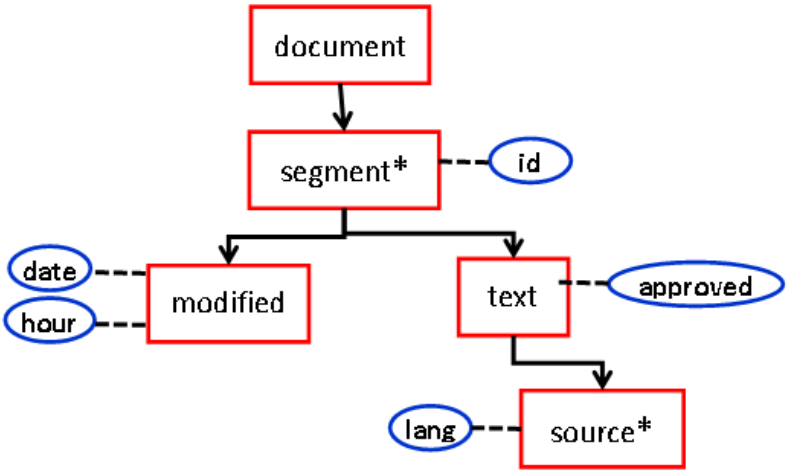

Figura 3.9: Árbol de un documento XML.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <document>
3      <segment id="277">
4          <modified date="2011-12-31" hour="18:52:15">EULA</modified>
5          <text approved="yes">
6              <source lang="enGB">
7                  Translation Teams for Joomla!1.5 may have also released fully localised
8                  installation packages where site, administrator and sample data are in the local language.
9              </source>
10             <source lang="esES">
11                 Es posible que los Equipos de Traducción de Joomla! 1.5 hayan localizado paquetes de instalación
12                 en los que el sitio web, el administrador y los datos de ejemplo estén en el idioma local.
13             </source>
14         </text>
15     </segment>
16       <segment id="277">
17         <modified date="2011-12-31" hour="18:52:15">USAL</modified>
18         <text approved="no">
19             <source lang="enGB">
20                 Translation Teams for Joomla!1.6 may have also released fully localised
21                 installation packages where site, administrator and sample data are in the local language.
22             </source>
23             <source lang="esES">
24                 Es posible que los Equipos de Traducción de Joomla! 1.6 hayan localizado paquetes de instalación
25                 en los que el sitio web, el administrador y los datos de ejemplo estén en el idioma local.
26             </source>
27         </text>
28     </segment>
29     <segment id="2232">      (...)
```

Figura 3.10: Fragmento del documento XML esquematizado en la Figura 3.9.

Tal y como se puede apreciar en la Figura 3.10, en la que se muestra parte de este documento, el texto traducible se encuentra en los nodos «source», por lo que supongamos que queremos seleccionar a todos los nodos con este nombre. Como ya hemos visto antes, tenemos dos opciones: **/document/segment/text/source** o, simplemente, **//source** (ver Figura 3.11).

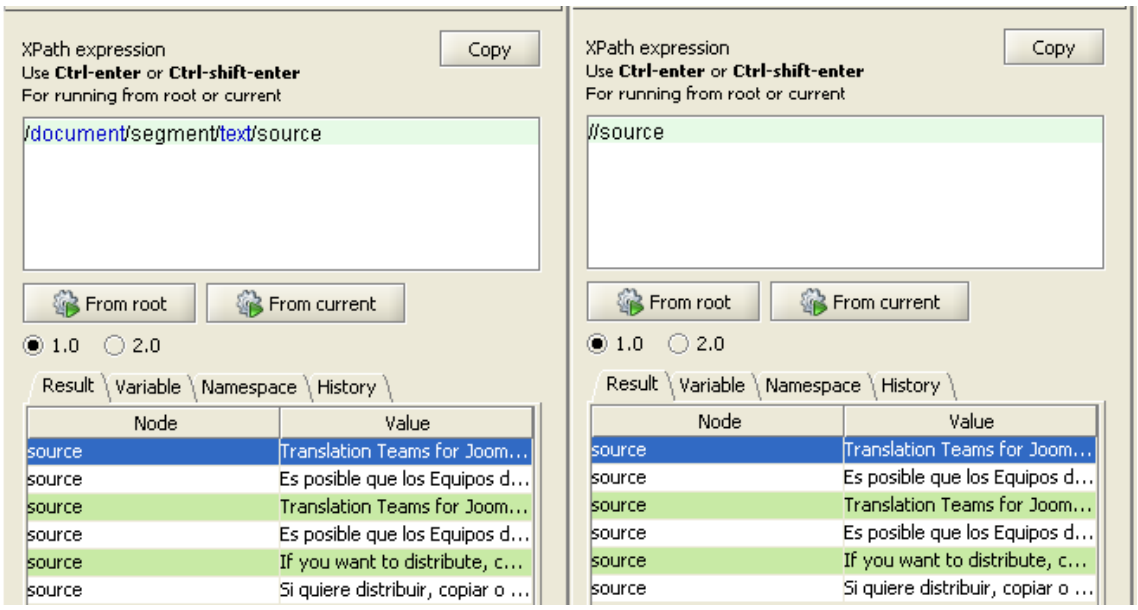

Figura 3.11: Mismos resultados en EditiX de dos expresiones XPath distintas.

Sin embargo, tal y como queda demostrado en la Figura 3.11, con esta expresión estamos seleccionando dos nodos que contienen el mismo texto en dos idiomas distintos. Si regresamos a la Figura 3.10 podemos ver que cada nodo «source» tiene un atributo «lang» cuyo valor es o bien «enGB» o bien «esES». Imaginemos que, en el supuesto que estamos planteando, sólo queremos seleccionar los nodos «source» cuyo atributo «lang» sea «esES».

Como ya hemos visto antes, para seleccionar un atributo se emplea el símbolo **@** seguido del nombre del mismo. No obstante, en este caso nuestro propósito no es simplemente el de dirigirnos unos atributos determinados, sino que queremos llegar a atributos con un valor concreto, por lo que la expresión XPath ha de ser más compleja.

Para seleccionar un atributo con un valor determinado, es preciso especificar el nombre del mismo (en ese caso, **@lang**) y, a continuación, un símbolo de igual seguido del nombre de los atributos que deseamos obtener entre comillas. En este ejemplo, el

valor que nos interesa es «esES», por lo que la expresión que nos lleva hasta dichos atributos es **@lang=”esES”**.

Pero la expresión de XPath aún no está completa, puesto que antes de mencionar el atributo hay que señalar a qué nodos está asociado. Para marcar esta unión, el lenguaje XPath se sirve de los corchetes, por lo que en el supuesto con el que estamos trabajando, tendríamos que escribir **//source[@lang=”esES”]** (ver Figura 3.12).

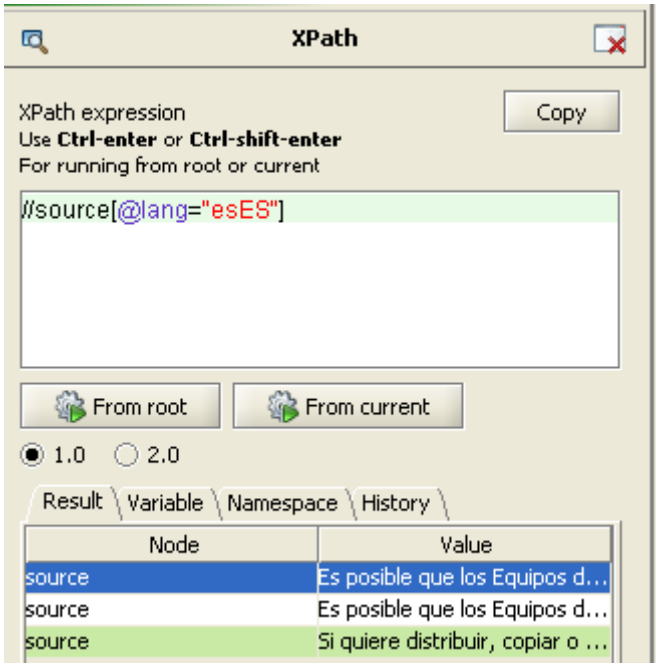

Figura 3.12: Resultados en EditiX de la expresión XPath //source[@lang=”esES”].

Para complicarlo un poco más imaginemos que, además de querer seleccionar sólo los nodos «source» cuyo atributo «lang» sea «esES», deseamos restringir más nuestra búsqueda. Como puede verse en la Figura 3.9, los elementos «text», padres de los «source», también tienen asociados atributos con distintos valores.

Supongamos que deseamos llegar sólo a aquellos elementos «source» en los que el valor del atributo «approved» de sus padres «text» sea «yes». Siguiendo la lógica del

lenguaje XPath que ya hemos aprendido, la expresión necesaria para seleccionar sólo los nodos «text» cuyo atributo «approved» sea «yes» es **//text[@approved=”yes”]**.

Tras esto, nos queda por especificar que únicamente queremos seleccionar los hijos de «text» llamados «source» cuyo atributo «lang» sea «esES». Para ello, escribiremos a continuación la expresión que ya hemos utilizado en el ejemplo de la Figura 3.12, pero eliminando una de las barras iniciales. De este modo, la expresión final sería **//text[@approved=”yes”]/source[@lang=”esES”]** (ver Figura 3.13).

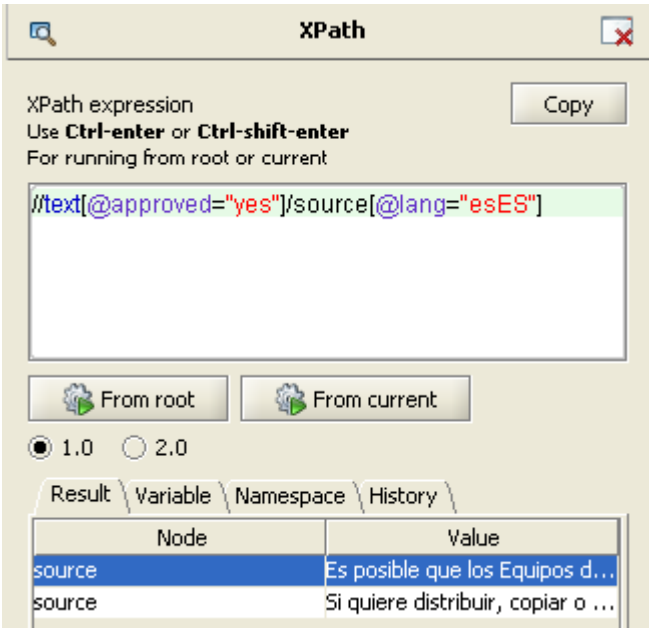

Figura 3.13: Resultados en EditiX de la expresión XPath //text[@approved=”yes”]/source[@lang=”esES”].

Para terminar, imaginemos que queremos (o debemos) rizar más el rizo y que, además de todas las condiciones especificadas en los ejemplos que se muestran en las Figuras 3.12 y 3.13, deseamos seleccionar sólo los fragmentos que han sido modificados por alguien en concreto. Como se muestra en las Figuras 3.9 y 3.10, cada nodo «segment» tiene un nodo hijo llamado «modified», en los que se especifica quién es el autor del último cambio.

Por ejemplo, supongamos que ahora nuestra intención es delimitar más la búsqueda presentada en la Figura 3.13 y que queremos seleccionar únicamente el nodo «source» que ha sido modificado (recordemos una vez más que esto aparece especificado en el nodo «modified») por «USAL».

Para ello, tendremos que añadir al comienzo de nuestra expresión **//segment[modified=”USAL”]**. En este caso no ponemos el símbolo de arroba delante de «modified» ya que, al contrario que en los casos anteriores, la información que necesitamos aparece especificada dentro del propio nodo y no en un atributo del mismo. De esta manera, la expresión final sería **//segment[modified=”USAL”]/text[@approved=”yes”]/source[@lang=”esES”]** (ver Figura 3.14)**.**

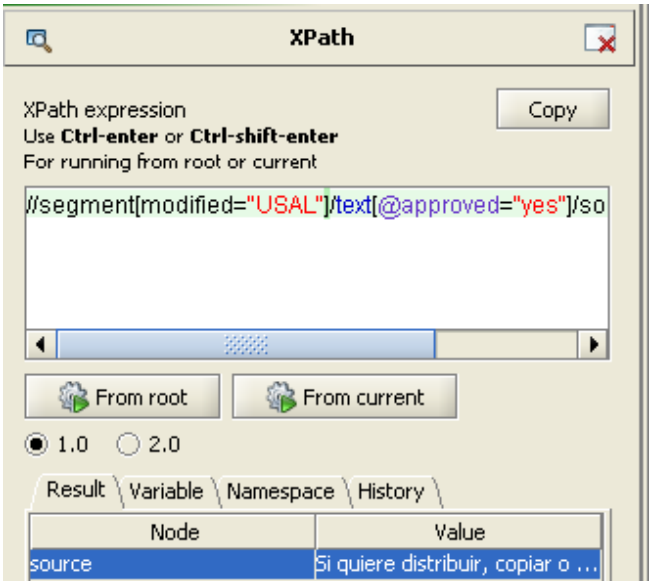

Figura 3.14: Resultados en EditiX de la expresión XPath //segment[modified=”USAL”]/text[@approved=”yes”]/source[@lang=”esES”].

4. Construcción de filtros XML en SDL Trados 2009

4.1. Introducción

Como ya hemos explicado en la introducción de este Trabajo de Fin de Grado, en el año 2008 el consorcio OASIS creó un estándar llamado XLIFF. Su propósito era facilitar la tarea de localización mediante un formato que separara el contenido de la estructura de los documentos, independientemente de cuál sea el formato de origen de estos.

Asimismo, también hemos explicado que las herramientas TAO tienen incorporados filtros que convierten cualquier documento al estándar XLIFF. No obstante, como sabemos, en ocasiones el traductor se ve obligado a modificar dichos filtros o crear los suyos propios para ajustarlos a los requerimientos específicos de un archivo concreto.

El propósito de este apartado es proporcionar unas directrices básicas que ayuden al traductor a entender el proceso de creación de filtros para documentos XML en SDL Trados Studio 2009. Una vez entendidas y asimiladas, el traductor podrá aplicar sus conocimientos a cualquier archivo XML que tenga que traducir en el ejercicio de su profesión.

4.2. El filtro estándar en SDL Trados Studio 2009

Como ya hemos dicho, SDL Trados Studio 2009 dispone de un filtro estándar para los ficheros XML. Antes de proseguir con este apartado, resulta preciso explicar brevemente parte de la lógica de SDL, pues resulta determinante a la hora de crear nuestros propios filtros personalizados. El programa establece, por defecto, que todo el texto contenido en los nodos es texto traducible. Del mismo modo, si no se le indica lo contrario, determina que el texto presente en los atributos es texto no traducible (SDL, 2009).

Por este motivo, cuando disponemos de un fichero en el que parte o todo el texto que debemos traducir se encuentra en los atributos, tenemos que especificarlo en nuestro filtro. Ocurre exactamente lo mismo cuando no tenemos que traducir todo el texto contenido en los nodos de nuestro documento XML: tendemos que especificárselo a nuestra herramienta TAO para que no nos muestre las partes del documento que no deseamos traducir.

4.3. Selección de ejemplos

Para ilustrar este apartado, hemos seleccionado una serie de documentos XML que cubren todo el abanico de necesidades o problemas que pueden surgirle a un traductor en el ejercicio de su profesión. De esta manera, las explicaciones que presentamos a continuación ofrecen una muestra de los diferentes tipos de filtro que éste puede precisar. Por supuesto, se trata de casos concretos que el profesional deberá adaptar a los documentos específicos con los que esté trabajando.

4.4. Ejemplo 1: seleccionar un subconjunto de nodos

Para empezar, vamos a imaginar que un cliente nos ha proporcionado el documento XML ilustrado parcialmente en la Figura 4.1. En esta imagen, hemos encuadrado en rojo los nodos del mismo que debemos traducir. Del mismo modo, se encuentran señalados en verde aquéllos que no tenemos que modificar.

Figura 4.1: Archivo XML del Ejemplo 1.

Como ya hemos explicado anteriormente, el filtro estándar de SDL Trados Studio 2009 muestra como traducible todo el texto presente en los nodos de los documentos XML. Sin embargo, en este caso concreto sólo deseamos traducir el que se encuentra en los nodos «value», por lo que tendremos que crear un filtro que satisfaga nuestras necesidades.

De no hacerlo, el programa nos presentaría todo el texto contenido en el documento, tal y como se muestra en la Figura 4.2, en la que aparecen enmarcados en verde los segmentos que no deseamos traducir.

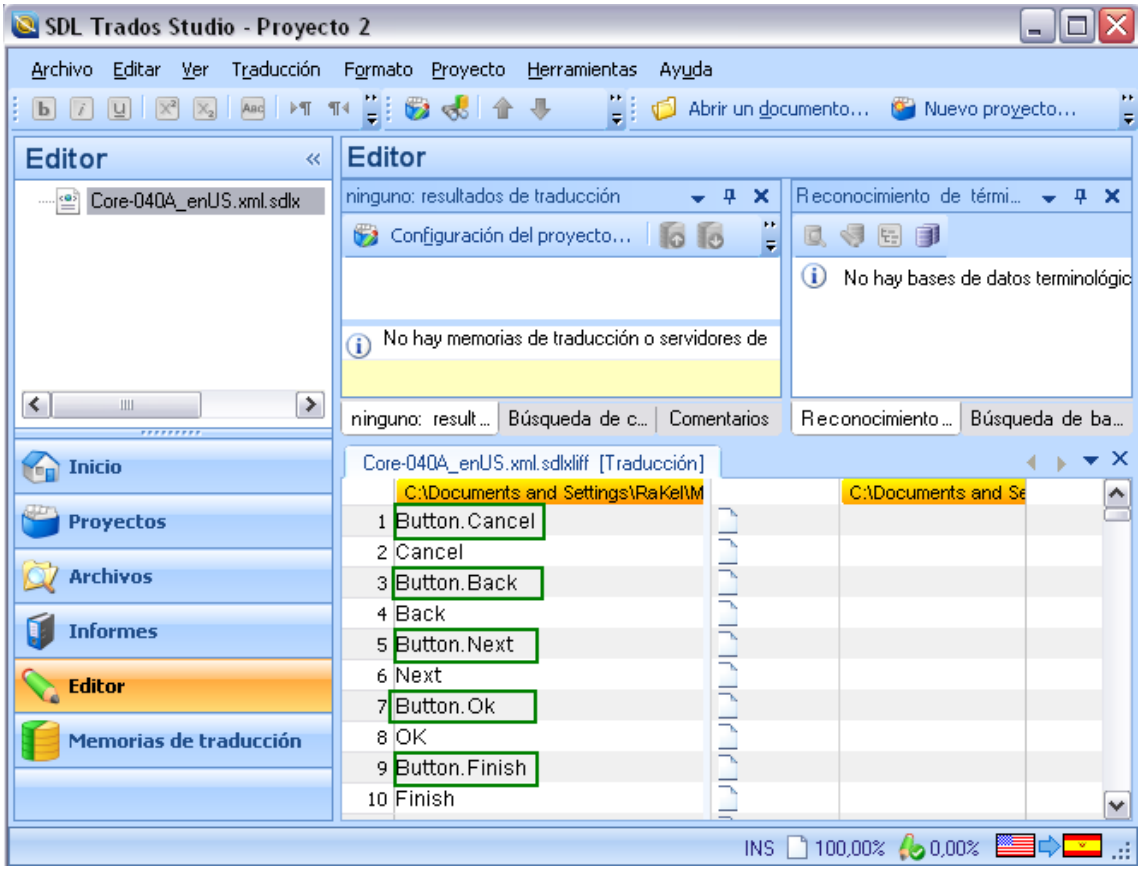

Figura 4.2: Documento del Ejemplo 1 con el filtro estándar de SDL.

Por esta razón, tendremos que crear un filtro que le diga a SDL que no queremos traducir ciertos fragmentos de nuestro archivo. Así, siguiendo la lógica del lenguaje XPath aprendida, tendremos que introducir en nuestra herramienta TAO una regla que diga que **//value** es «Siempre traducible». Con esto, como ya sabemos, le estamos indicando que el texto contenido en todos los nodos «value» del documento son «Siempre traducibles».

Además de esto, tendremos que indicarle a SDL que el resto de nodos son «No traducibles», para que no nos los muestre. La expresión que precisamos para este fin es **//*** igual a «No traducible», pues con ella seleccionamos a todos los nodos.

Antes de proseguir con nuestra explicación, es preciso aclarar una característica de SDL Trados Studio 2009: para este programa, el orden en que se introducen las

expresiones XPath al crear un filtro condiciona el modo en que se aplican. Es decir, que la expresión colocada encima prima sobre las que se encuentran por debajo de ésta.

Esto es importante, porque muchas veces una de las expresiones introducidas incluye a parte de las otras, y el orden en que se dispongan puede hacer que SDL no nos muestre los fragmentos del documento que queremos traducir. Por ejemplo, en la Figura 4.2 aparece representada de forma esquemática una fracción del documento con el que estamos trabajando ahora.

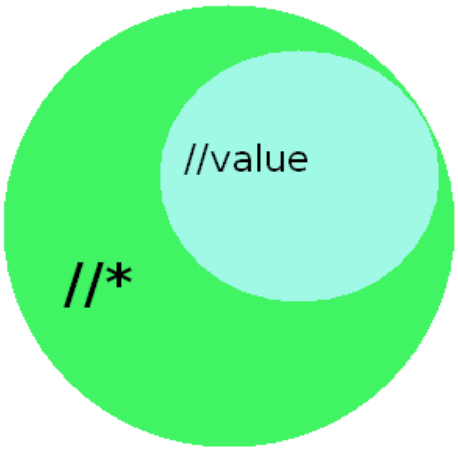

Figura 4.3: Esquema básico del documento XML del Ejemplo 1.

El círculo verde representa la expresión **//*** que, como ya sabemos, se refiere a todos los nodos del documento. Dentro de éste, el círculo azul contiene los nodos que se encuentran tras la expresión **//value** o, en otras palabras, todos los nodos «value» del fichero.

De esta manera, si primero le indicáramos a SDL que **//*** es «No traducible» y, a continuación, que **//value** es «Siempre traducible», el editor de la herramienta TAO no nos mostraría nada de texto, porque la primera orden prima sobre la segunda (ver Figura 4.4).

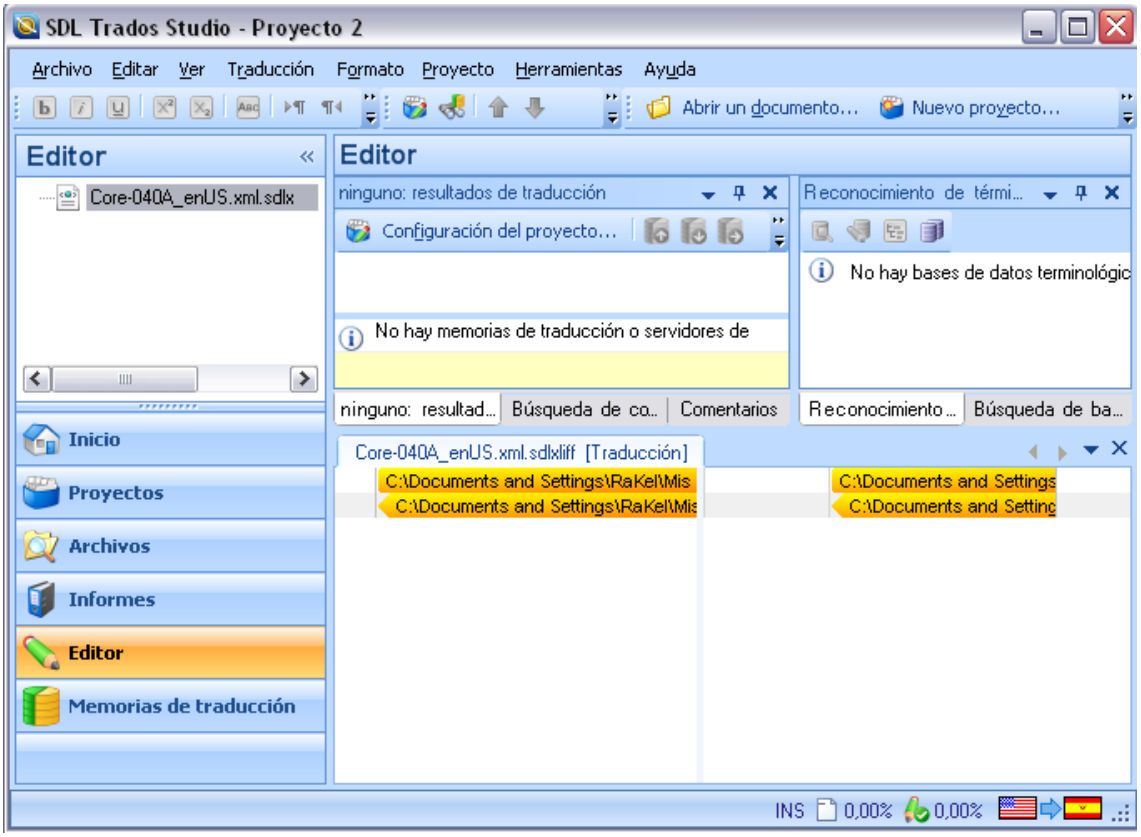

Figura 4.4: Resultado de introducir las reglas en el orden equivocado en el filtro del Ejemplo 1.

Por este motivo, el orden debe ser el contrario. Primero, debemos decirle a SDL que **//value** es «Siempre traducible» y, después, aclararle que no queremos que nos muestre el resto de los nodos (o, lo que es lo mismo, que //* es «No traducible»). El resultado sería el que se muestra en la Figura 4.5, en la que se puede ver el documento original, el editor de SDL una vez aplicado nuestro filtro y el documento de destino.

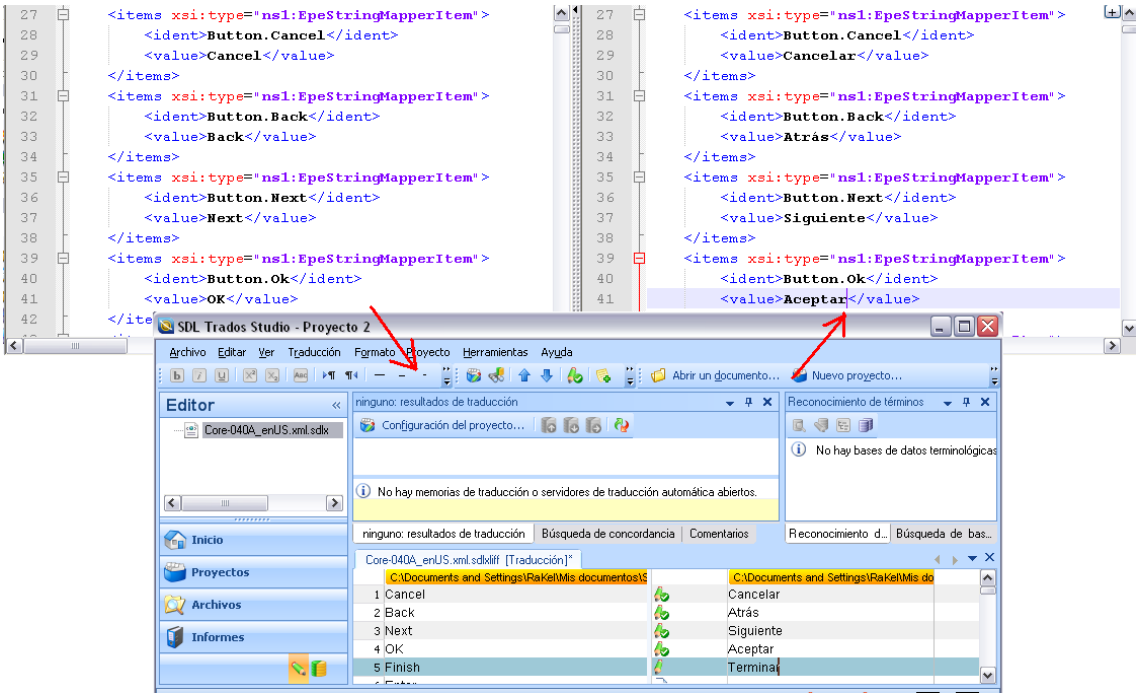

Figura 4.5: Resultado del filtro bien creado para el documento del Ejemplo 1: documento origen, archivo bilingüe y documento de destino.

4.5. Ejemplo 2: seleccionar un subconjunto de atributos

El fichero con el que vamos a trabajar en esta ocasión ya nos resulta familiar, pues lo empleamos para comprender los entresijos del lenguaje XPath. Se trata en concreto del mostrado en la Figura 3.4, perteneciente al programa NotePad++. Después de revisar todo el archivo XML, hemos delimitado en qué partes del documento se encuentra el texto que debemos traducir.

En la Figura 4.6 aparece representada la estructura en árbol de nuestro documento XML. En ella están resaltados en celeste aquellos atributos que contienen el texto traducible y, por consiguiente, los que debemos seleccionar con nuestro filtro. Como se puede ver (y como ya dijimos en el apartado anterior), en este archivo todo el texto traducible se encuentra ubicado en los atributos.

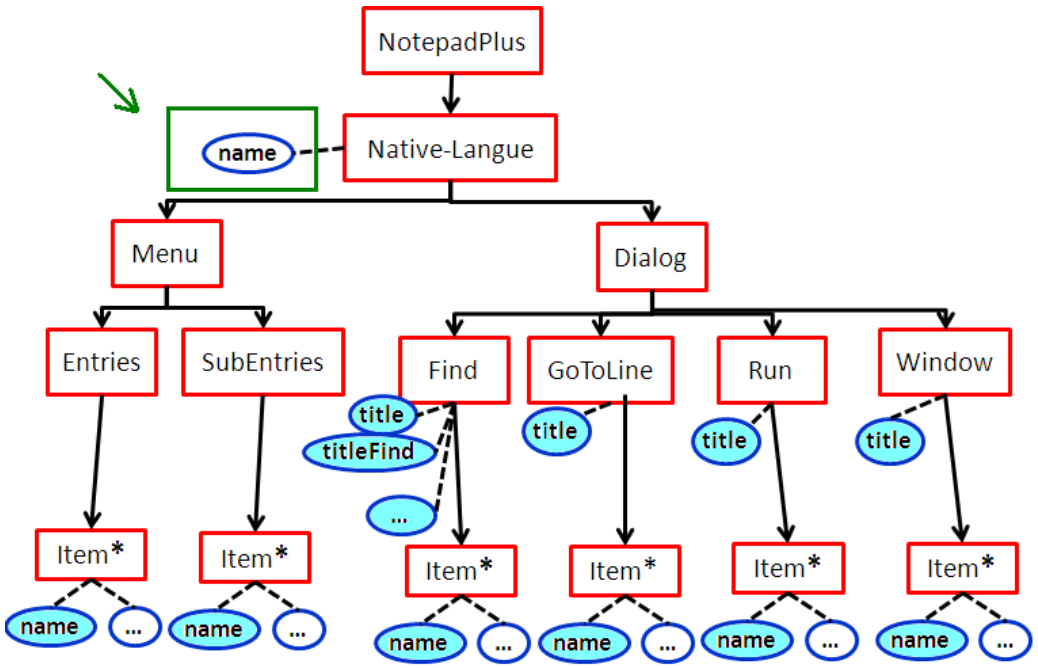

Figura 4.6: Árbol del documento XML del Ejemplo 2.

En concreto, nos interesan todos los atributos «name» del nodo «Item»; todos los atributos «title» del documento; y todos los atributos asociados al nodo «Find», independientemente de cuál sea su nombre. Puesto que en todos estos casos, como ya hemos explicado, SDL establece que se trata de texto no traducible, tendremos que servirnos del lenguaje XPath para crear un filtro que nos muestre los segmentos que deseamos traducir.

Así, para nuestra primera selección, como ya explicamos en el apartado anterior, nos valdremos de la expresión XPath **//Item/@name** (ver Figura 3.6). No podemos servirnos de **//@name** para seleccionar todos los atributos del documento con ese nombre porque, como puede observarse en la Figura 4.6, el nodo «Native Language» también contiene un atributo con el mismo nombre, que no queremos traducir.

Para el segundo caso, como también vimos en el apartado anterior (ver Figura 3.7), emplearemos la expresión **//@title**. Por último, para seleccionar todos los atributos del nodo «Find», utilizaremos la expresión **//Find/@*** (ver Figura 3.8).

En la Figura 4.7 se puede ver la pantalla de creación de nuestro filtro, con las reglas introducidas, y el editor de SDL una vez aplicado éste al documento con el que estamos trabajando.

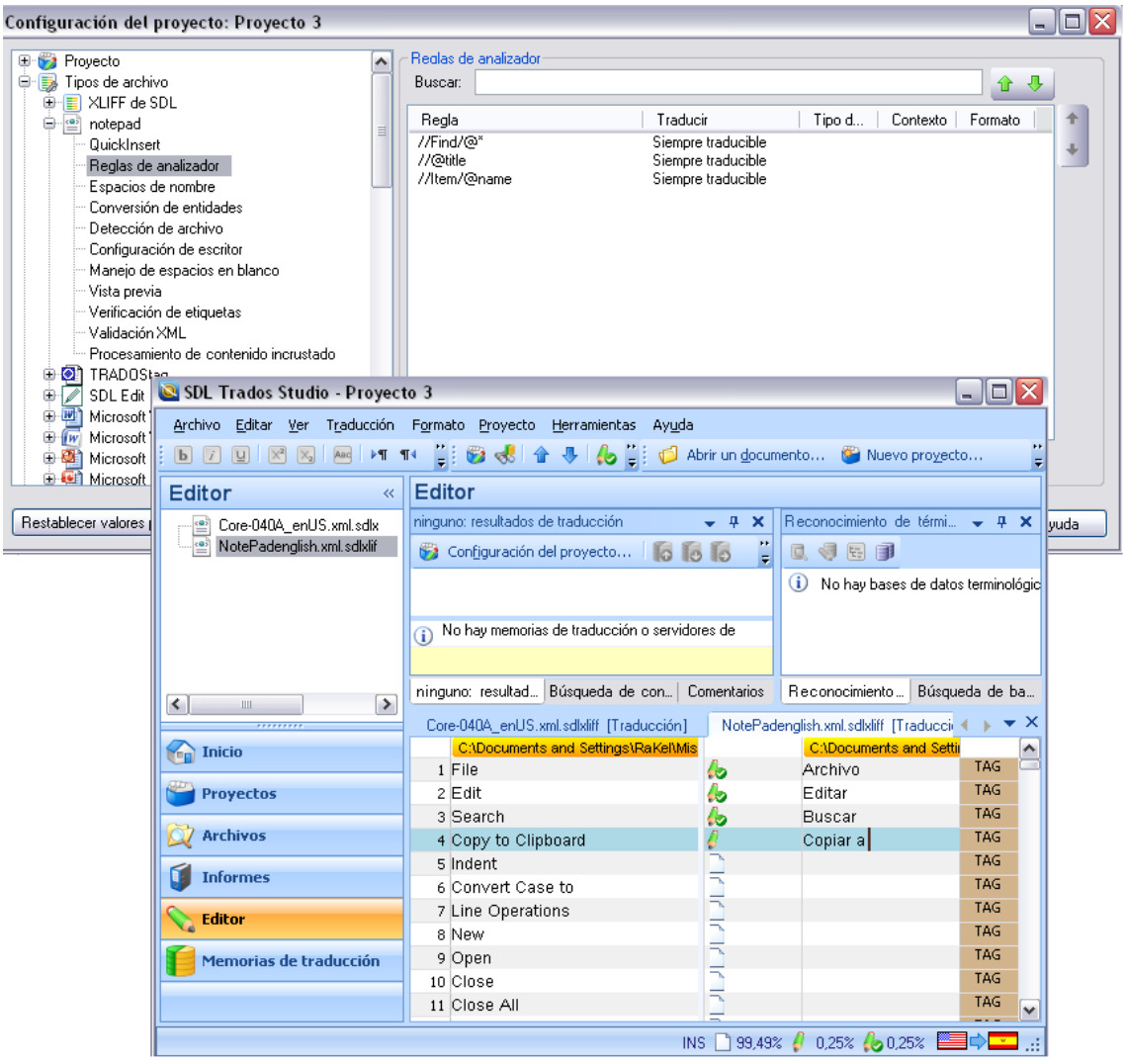

Figura 4.7: Filtro para seleccionar atributos del Ejemplo 2 y editor de SDL.

4.6. Ejemplo 3: seleccionar nodos que satisfagan unas condiciones

Ahora, vamos a suponer que un cliente nos ha proporcionado el documento XML parcialmente representado en la Figura 4.8.

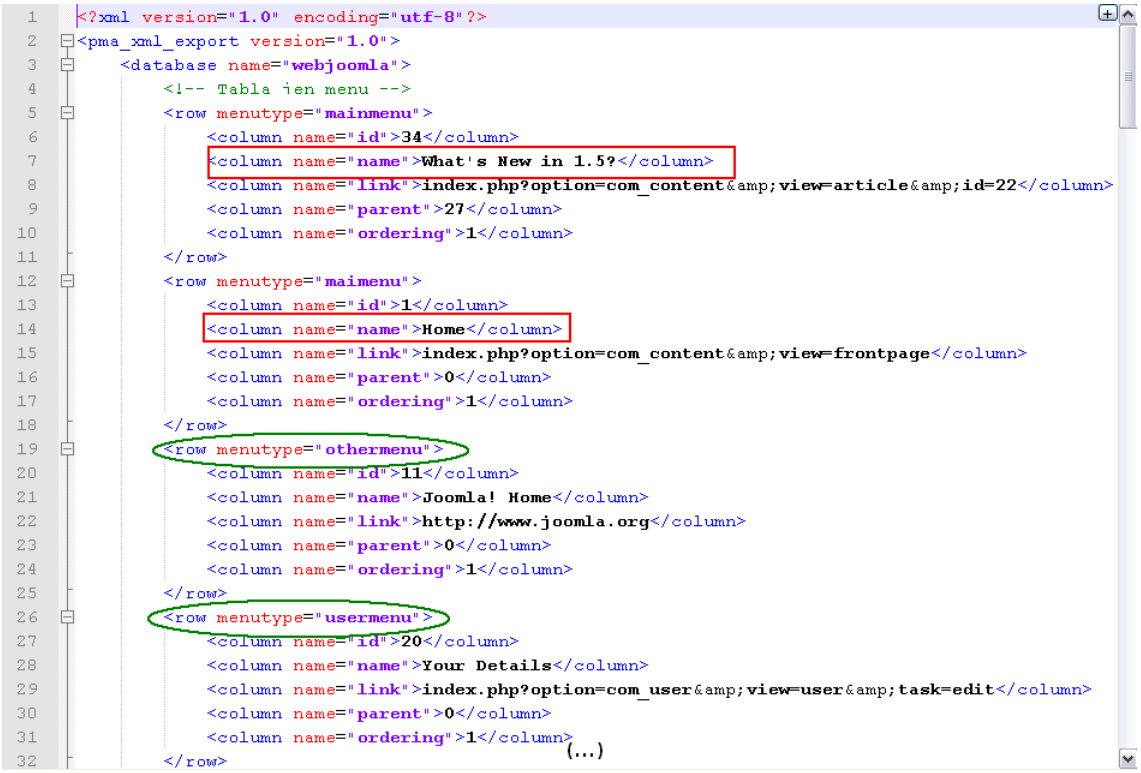

Figura 4.8: Documento XML del Ejemplo 3.

De él, nos dice, hemos de traducir exclusivamente los nodos «column» hijos de «row» cuyo atributo «name» sea «name», pero sólo cuando el atributo «menutype» de «row» sea igual a «mainmenu». Es decir, tenemos que traducir el texto que en la Figura 4.8 aparece enmarcado en rojo. Es muy importante tener en mente que sólo deseamos seleccionar esos nodos cuando el atributo «menutype» de «row» sea igual a «mainmenu» porque, como se puede ver en la Figura 4.8 señalado en verde, dicho atributo tiene distintos valores a lo largo del documento.

Para delimitar nuestra selección, siguiendo la lógica aprendida en el apartado anterior, la expresión que debemos emplear es **//row[@menutype=”mainmenu”]/column[@name=”name”]**. No obstante, hay que recordar que, como ya hemos dicho anteriormente, SDL muestra por defecto el texto contenido en todos los nodos. Por este motivo, además de especificar en nuestro filtro qué nodos queremos traducir, también tenemos que determinar cuáles no.

De esta manera, en primer lugar introduciremos la expresión anterior, en la que indicaremos que el texto que contiene es «Siempre traducible». Tras esto, le diremos a SDL que no queremos que nos muestre el texto contenido en el resto de nodos o, lo que es lo mismo, que **//*** es «No traducible».

El resultado será el que se muestra en la Figura 4.9, en la que se puede ver tanto el orden de las normas introducidas como los segmentos de texto que presenta el editor de SDL una vez aplicado el filtro.

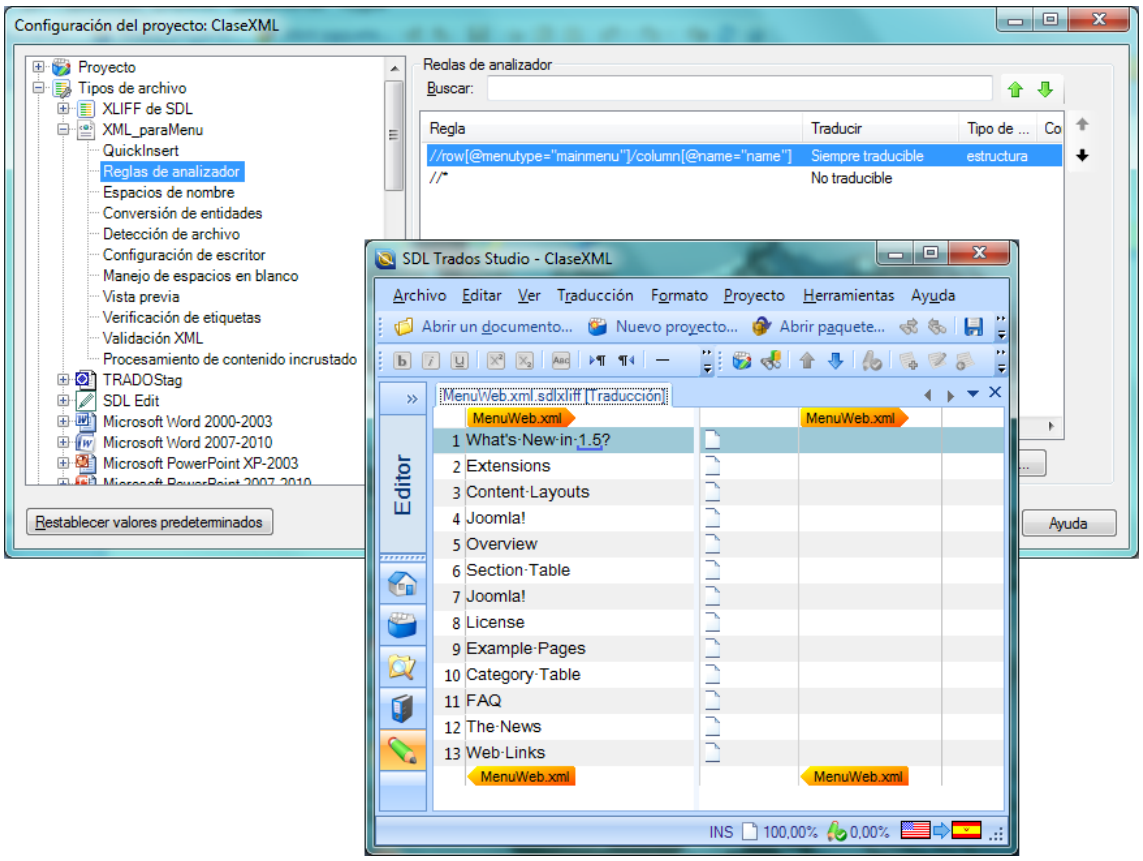

Figura 4.9: Filtro para el documento del Ejemplo 3 y editor de SDL.

4.7. Ejemplo 4: nodos de estructura y nodos internos

Por último, imaginemos que nos han entregado el documento representado en la Figura 4.10, del cual sólo tenemos que traducir la parte enmarcada en rojo. Ya hemos trabajado con un fichero similar a este en el apartado anterior (ver Figura 3.10); la diferencia es que en el que tenemos ahora todo el texto está en inglés, a la espera de ser traducido.

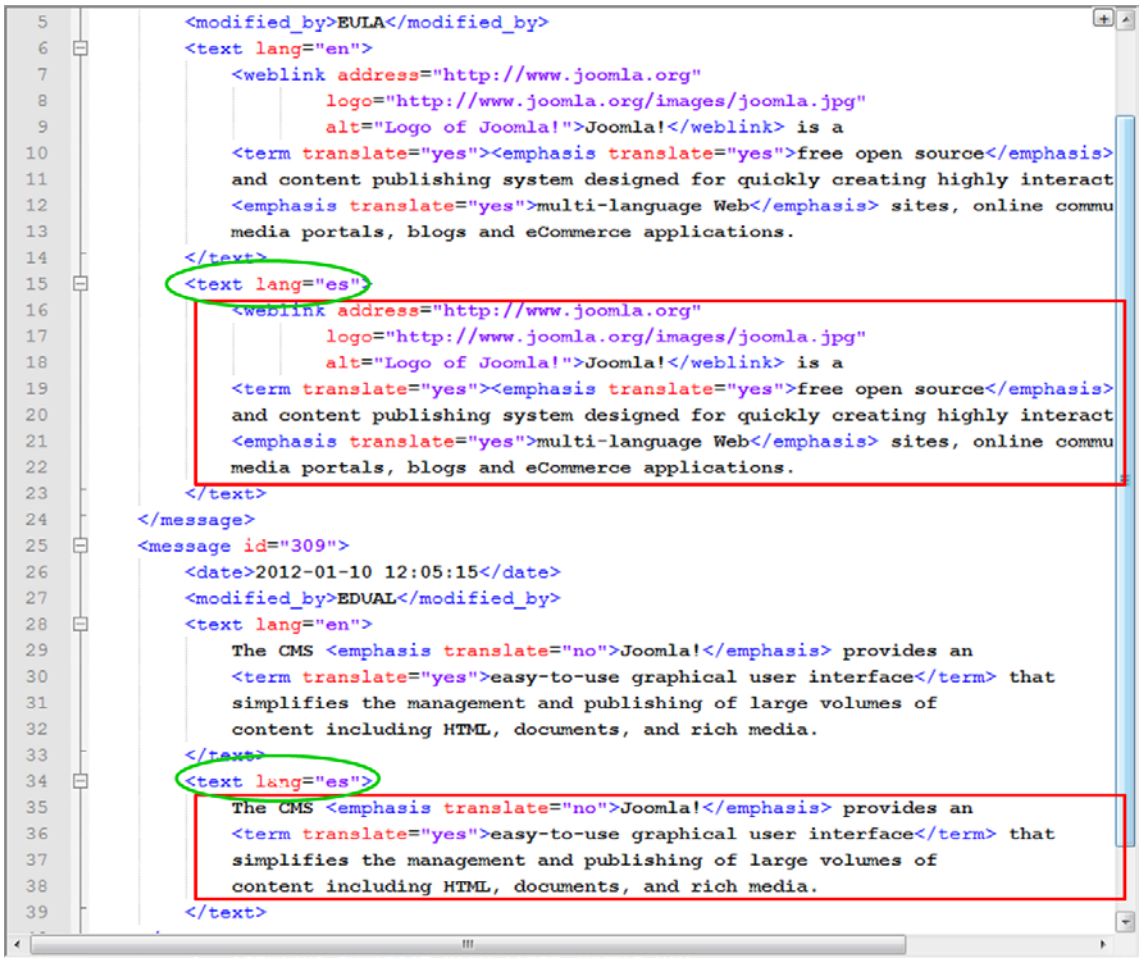

Figura 4.10: Documento XML del Ejemplo 4 con etiquetas internas.

Observemos la Figura 4.10 con más detalle. En principio, hay que extraer para traducir sólo el texto de los nodos «text» cuyo atributo «lang» tenga valor «es». Es decir, los elementos que en la ilustración están enmarcados con una elipse en verde. Como ya sabemos, para ello nos bastaría con indicar que **//text[@lang=”es”]** es «Siempre traducible», mientras que el resto de nodos son no «No traducible».

Sin embargo, vemos que dentro de los nodos «text», entre el propio texto, aparecen también una serie de etiquetas (por ejemplo, entre otras, podemos ver «emphasis», «term»…) que, en este caso, deben tratarse como «internas» al propio segmento. Es decir, el traductor deberá llevar a la lengua meta el texto del nodo «text» intercalando, en el lugar apropiado, las etiquetas que contenga. Nótese que vamos a

crear el filtro con la suposición de que no conocemos cómo se llaman estos nodos descendientes de «text».

No obstante, supongamos que aunque desconozcamos a priori los nodos descendientes de «text», sí que sabemos que algunos de ellos pueden contener un atributo llamado «translate» cuyo texto no se deberá traducir cuando tengan asociado el valor «no». En la figura anterior, pueden observarse dos apariciones de dicho atributo, una vez con valor «yes» y otro con valor «no».

Por tanto, concretando, nuestro filtro deberá:

1. Extraer el texto, como segmentos, sólo de los nodos «text» cuyo atributo «lang» tenga el valor «es». Para ello, utilizaremos la expresión **//text[@lang=”es”]**.

2. Mantener como etiquetas internas todos los elementos descendientes de los nodos anteriores. Esto podemos concretarlo en XPath de la siguiente forma: **//text[@lang=”es”]//***

3. No permitir la traducción del texto de aquellos nodos que tengan un atributo «translate» con el valor «no». En este caso, usaremos la expresión **//text[@lang=”es”]//[translate=”no”]**.

4. Finalmente, como ya hemos hecho con los otros filtros, tenemos que indicarle a SDL que el contenido del resto de nodos (//*) es «No traducible».

Puesto que, como ya hemos dicho anteriormente, en la creación de los filtros el orden de las reglas determina el resultado final, tendremos que tener cuidado con el modo en que organizamos nuestras expresiones. Así, //*, como ya sabemos, tendrá que ser la última, ya que de no ser así se superpondría a todas las demás.

En cuanto al resto de las órdenes, es preciso tener en cuenta que no queremos que SDL permita la modificación del texto de los nodos cuyo atributo «translate» sea «no». Esta norma, por tanto, ha de prevalecer al menos sobre la que indica que el texto de los nodos «text» cuyo atributo «lang» sea «es» es «Siempre traducible». En otras palabras, **//text[@lang=”es”]/*[@translate=”no”]** tiene que posicionarse por encima de **//text[@lang=”es”]** y de **//text[@lang=”es”]/***. En caso contrario, SDL nos mostraría como traducible un contenido que no deseamos traducir (ver Figura 4.11).

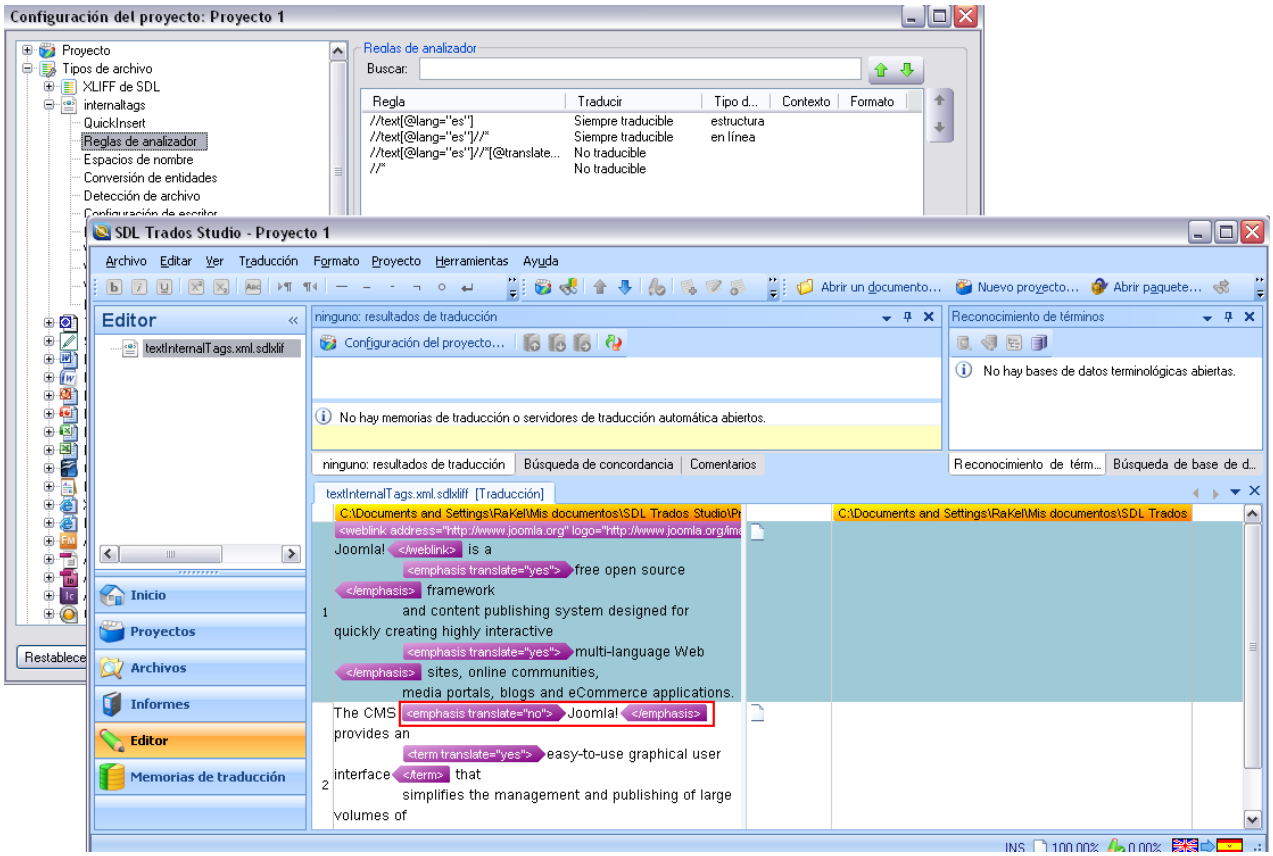

Figura 4.11: Filtro en SDL para el Ejemplo 4 con las reglas en el orden incorrecto: contenido «protegido» aparece como traducible.

Sin embargo, lo que queremos es que SDL nos muestre el texto de los nodos cuyo atributo «translate» sea igual a «no» como contenido protegido (ver Figura 4.12). Para ello, como acabamos de decir, en la jerarquía de órdenes debemos posicionar a

//text[@lang=”es”]//*[@translate=”no”] por encima del resto de reglas (ver Figura 4.13).

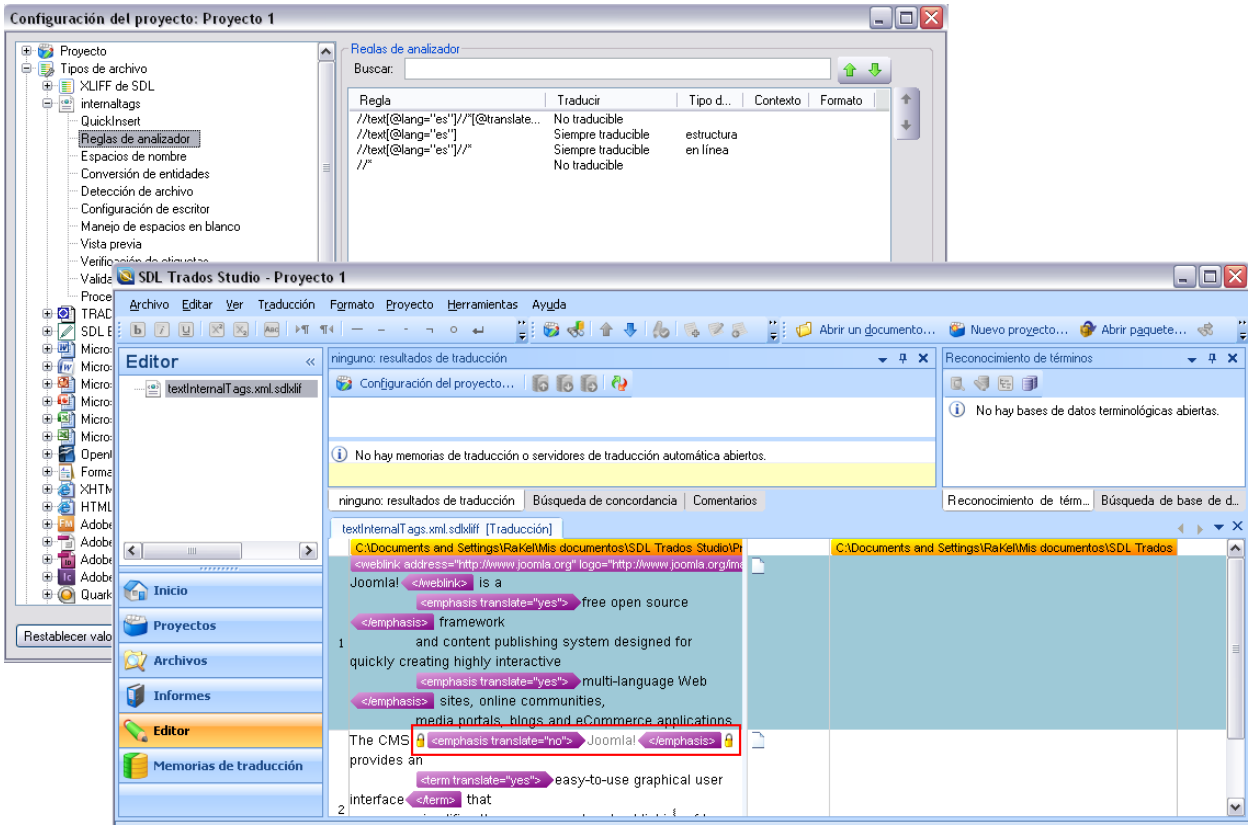

Figura 4.13: Filtro SDL para el Ejemplo 4 con las reglas en el orden correcto: contenido «protegido» aparece bloqueado.

Ya tenemos las reglas en el orden adecuado, pero el editor de SDL nos muestra el texto con unos espacios muy amplios entre palabras que hacen que la visión del texto resulte poco agradable. Por suerte, esta apariencia se puede modificar. Para ello, cuando introduzcamos en el filtro las reglas tendremos que seleccionar la opción «Normalizar al menos que xml:space=’preserve’» en la opción «Espacio en blanco» (ver Figura 4.14). Al seleccionar este parámetro, los espacios en el editor de texto aparecerán como en un texto normal (ver Figura 4.15).

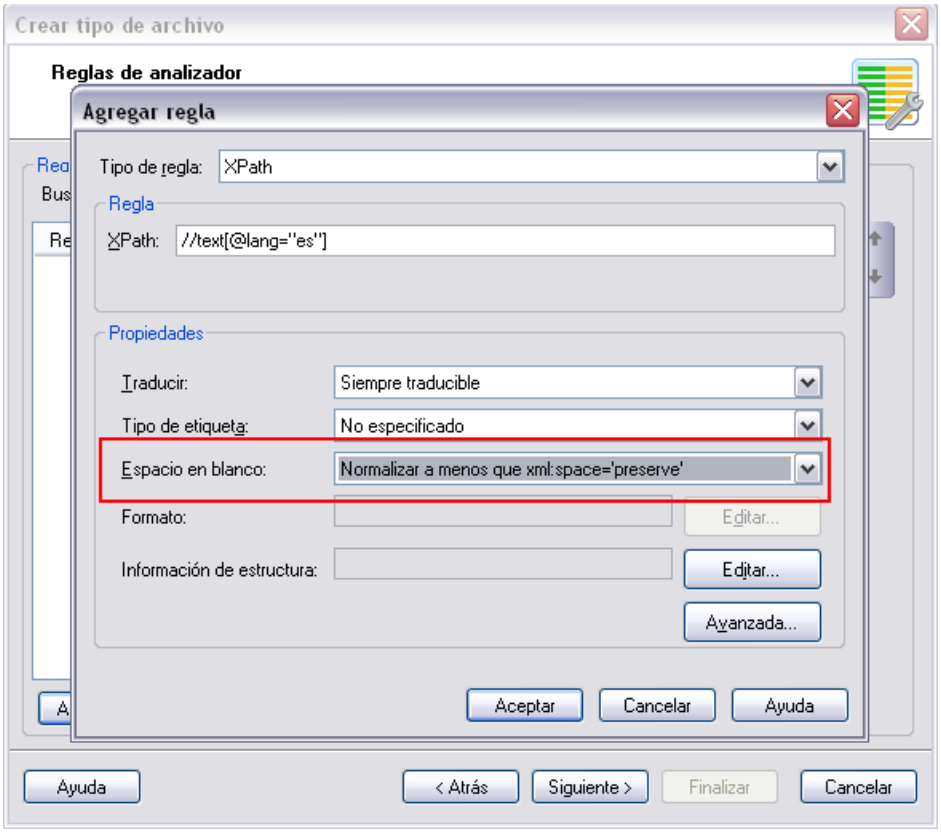

Figura 4.14: Menú de SDL para modificar la presentación de los espacios.

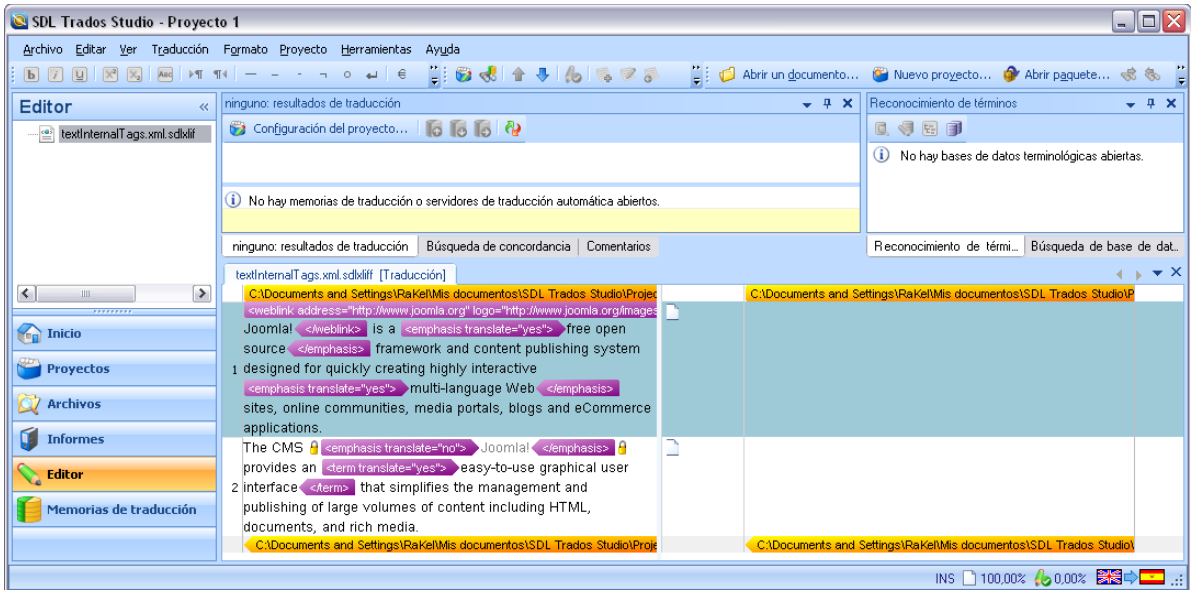

Figura 4.15: Vista en el editor del documento del Ejemplo 4 con los espacios normalizados.

5. Conclusiones

El presente Trabajo de Fin de Grado ha pretendido proporcionar a los traductores un primer acercamiento práctico a la forma de organizar documentos en lenguaje XML. Asimismo, también se ha centrado en realizar una breve introducción al lenguaje propuesto por la industria para la selección de subconjuntos concretos de cualquier documento: XPath.

Creemos haber podido demostrar que éste último (XPath), en un principio desconocido para el mundo de la traducción, no es tan complicado como podría parecer a simple vista. De hecho, mediante un número reducido de ejemplos prácticos, creemos haber conseguido familiarizar al traductor con el mismo, facilitándole la tarea para emplearlo en futuras ocasiones.

Asimismo, en el apartado relativo a los filtros en SDL Trados Studio 2009, consideramos que hemos proporcionado una aplicación práctica del lenguaje XPath que les resultará útil a los traductores. Nuestra intención ha sido, a través de los ejemplos seleccionados, cubrir una casuística lo más amplia posible para que pueda ser adaptada a otros documentos XML, sea cual sea su estructura u organización.

No obstante, la extensión restringida de este Trabajo de Fin de Grado sólo nos ha permitido proporcionar una guía práctica para un programa TAO concreto (SDL Trados Studio 2009). Se trata de un primer arranque que podría –y debería- completarse con el análisis de los filtros en otras herramientas del mismo tipo, tengan incorporado el lenguaje XPath a su funcionamiento interno o no.

Se trata de algo de suma importancia pues, como ya hemos mencionado con anterioridad, los incesantes cambios informáticos obligan al traductor o localizador a adaptarse sin descanso. Dada la existencia de unas herramientas que facilitan la tarea de estos profesionales, esta aclimatación constante debería comenzar precisamente por saber cómo sacar el máximo partido de ellas.

Bibliografía y recursos

GUTIÉRREZ Rodríguez, Abraham (2001): XML a través de ejemplos. Madrid: Ra-ma.

LECOMPTE, Sébastien y BOULANGER, Thierry (2009): XML práctico. Bases esenciales, conceptos y casos prácticos. Barcelona: Ediciones ENI.

MORADO Vázquez, Lucía y FILIP, David (2012): XLIFF Support in Cat Tools: Results of the survey. OASIS.
Disponible en: http://www.localisation.ie/resources/XLIFFSotAReport_20120210.pdf [Fecha de consulta: 19 de junio de 2012]

OASIS (2012). Advancing open standards for the information society.
Disponible en: https://www.oasis-open.org/. [Fecha de consulta: 21 de junio de 2012]

OASIS (2008): XLIFF 1.2 Specification.
Disponible en: http://docs.oasis-open.org/xliff/xliff-core/xliff-core.html [Fecha de consulta: 21 de junio de 2012]

RAYA, Rodolfo M. (2004): XML Localisation Interchange File Format as an intermediate file format. [en línea]
Disponible en: http://www.maxprograms.com/articles/xliff.html [Fecha consulta: 22 de Junio de 2012]

RIVAS, Santos (2001): Tutorial de XPath. [en línea]
Disponible en: http://geneura.ugr.es/~victor/cursillos/xml/XPath/ [Fecha de consulta 12 de mayo de 2012]

SDL (2009). On line help for SDL Trados Studio: Working with XML Files.

W3C (2008): Extensible Markup Language (XML) 1 (Fith Edition). [en línea]
Disponible en: http://www.w3.org/TR/REC-xml [Fecha de consulta: 11 de junio de 2012]

W3C (2012a): Extensible Markup Language. [en línea]
Disponible en: http://www.w3.org/XML/. [Fecha de consulta: 11 de junio de 2012]

W3C (2012b): W3C HTML. [en línea]
Disponible en: http://www.w3.org/html/ [Fecha de consulta: 11 de junio de 2012]

W3C (2011): W3C XML Path Language (XPath) 2.0 (Second Edition). [en línea]
Disponible en: http://www.w3.org/TR/xpath20/ [Fecha de consulta 13 de junio de 2012]

W3Schools (2012): XPath Tutorial. [en línea]

Disponible en: http://www.w3schools.com/xpath/default.asp [Fecha de consulta: 1 de junio de 2012]

ANEXO 1 – Interacción con el software EditiX: el procesador XPath empleado

Para explicar el apartado de este trabajo relativo al lenguaje XPath, hemos decidido emplear el procesador de archivos XML "EditiX". Algunos de los motivos que nos han llevado a elegirlo son que se trata de un programa multiplataforma (es decir, que hay versiones para los diferentes sistemas operativos de escritorio) y que la empresa (JAPISoft) distribuye una versión gratuita (*free*), limitada en funcionalidad, pero suficiente para nuestros propósitos, que puede descargarse en http://www.editix.com/.

Asimismo, resulta de gran utilidad a la hora de crear filtros, pues permite comprobar la validez de las expresiones XPath antes de introducirlas en la herramienta TAO con la que estemos trabajando. Esto es de suma importancia, pues si hemos errado en su formulación podremos corregirlo de inmediato, mientras que si nos damos cuenta del fallo cuando ya hemos creado el filtro, tendremos que empezar el proceso desde el principio.

Una vez dicho esto, creemos que es importante realizar una breve introducción sobre el funcionamiento básico de EditiX antes de comenzar con la explicación del lenguaje XPath. En primer lugar, se debe abrir el documento XML con el que se quiera trabajar. Para ello, existen dos opciones: *File > Open…* o seleccionar el icono que aparece enmarcado en rojo en la Figura A1.

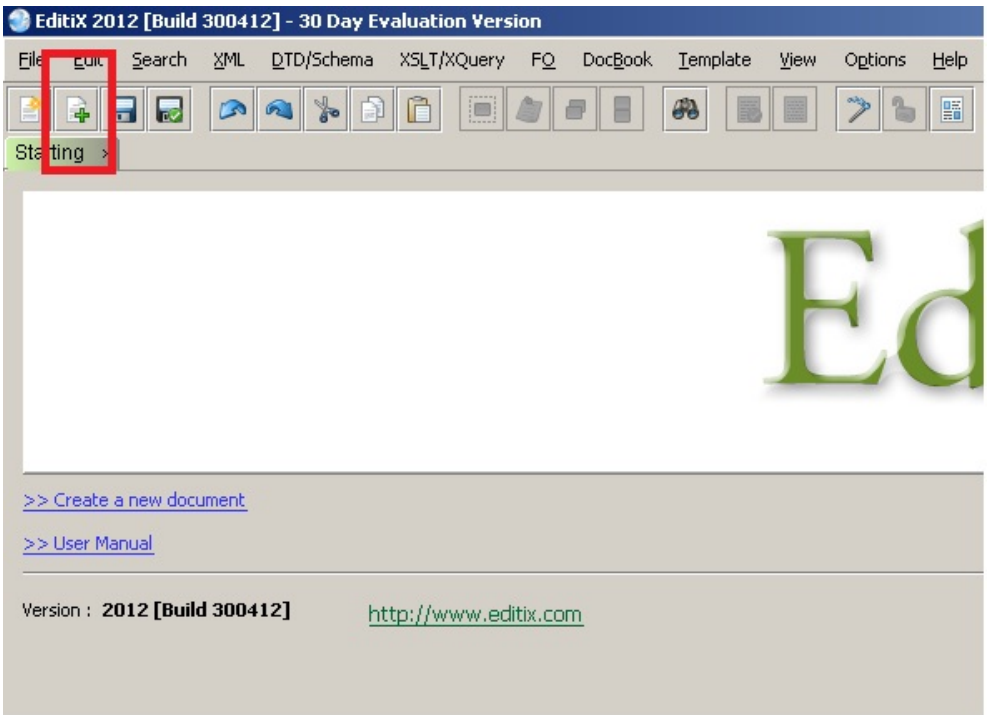

Figura A1: Cómo abrir un documento con EditiX.

Una vez hecho esto, tal y como se puede ver en la Figura A2, la pantalla queda dividida en tres partes. A la derecha, EditiX nos presenta el archivo XML con el que vamos a trabajar; en el centro, los nodos principales del documento; y, a la izquierda, el panel en el que debemos introducir nuestras expresiones XPath (ver Figura A4).

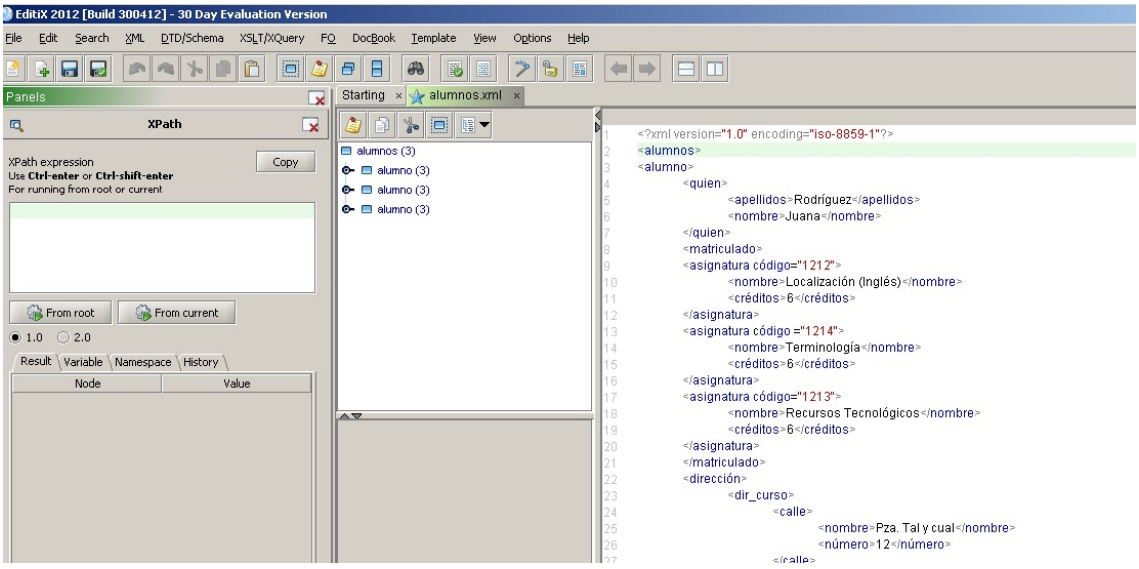

Figura A2: Vista de un documento XML abierto en EditiX.

Es posible que EditiX sólo nos muestre la pantalla de la derecha y la del centro. Si fuera así, lo único que tenemos que hacer es seleccionar *View* > *Windows* > *XPath view* para que nos aparezcan todos los paneles que necesitamos (ver Figura A3).

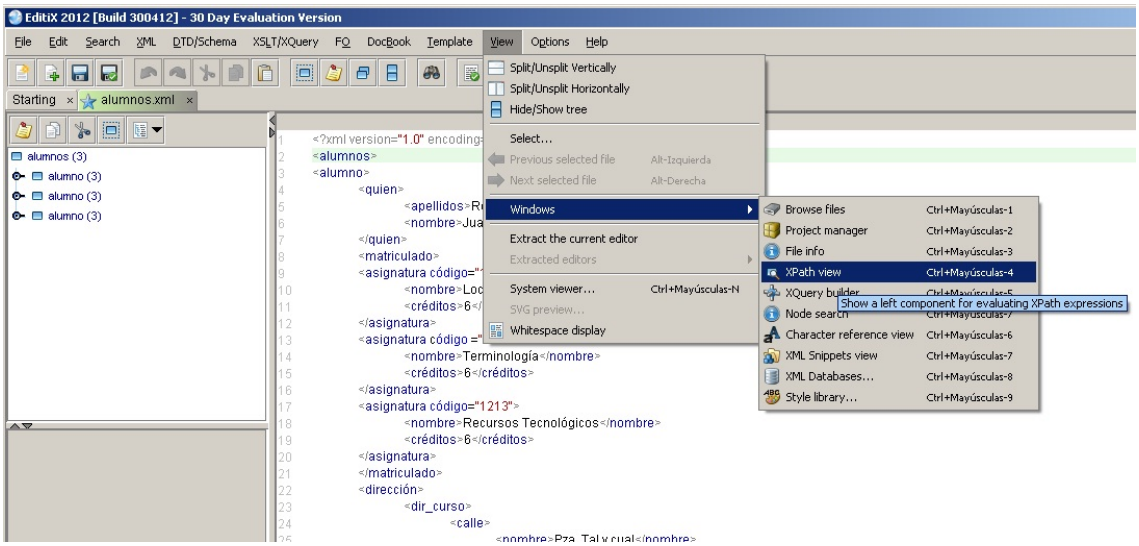

Figura A3: Cómo obtener el panel para introducir las expresiones XPath.

Tras introducir nuestra expresión XPath en el panel de la izquierda, tendremos que pulsar <Control> +< Intro> o el botón «From root» para que EditiX busque en todo el documento, empezando desde el nodo raíz. A continuación, si nuestra expresión es correcta, el programa nos mostrará debajo de ella los resultados que ha encontrado en el documento (ver Figura A4). En caso de que contenga algún error, nos lo hará saber (ver Figura A5) para que lo enmendemos.

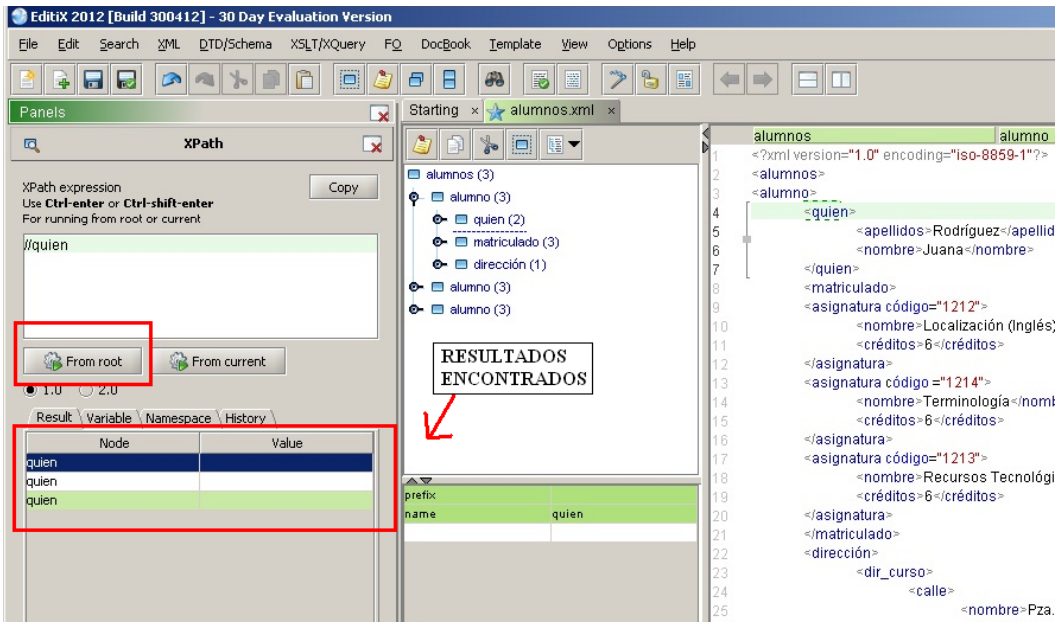

Figura A4: Introducir una expresión XPath y obtener resultados.

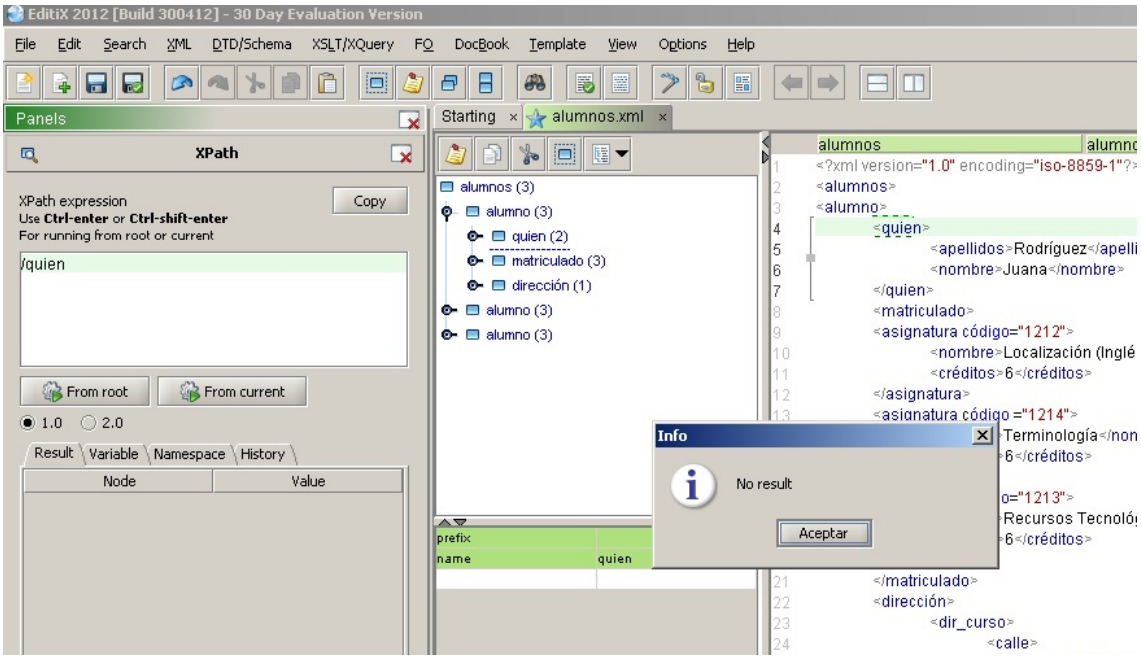

Figura A5: Mensaje de EditiX cuando nuestra expresión XPath no funciona.

Si nos movemos por los resultados que nos ofrece EditiX en el panel de la izquierda, el editor de XML nos irá señalando las partes del documento en que estos se encuentran. No obstante, para ilustrar los ejemplos de XPath que aparecerán a lo largo

de todo este apartado, presentaremos imágenes en las que se muestre, únicamente, el panel de EditiX con la expresión y los resultados (ver Figura A6).

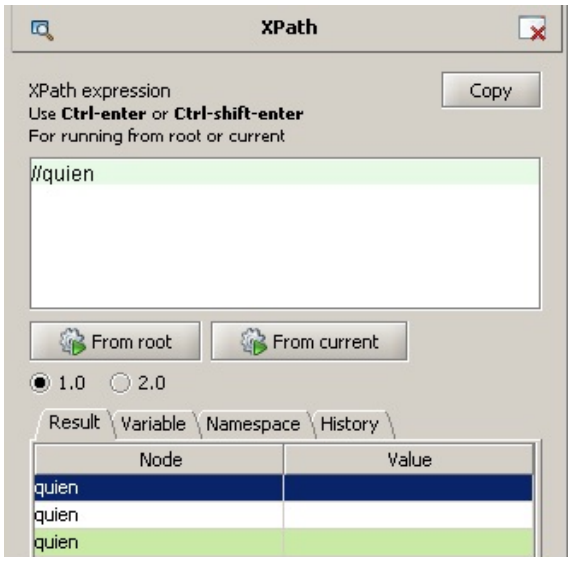

Figura A6: Ejemplo de pantalla de EditiX que se mostrará en el apartado de XPath.