Revista
Española de

Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 7, No. 1, abril, 2011

Web de la editorial: www.ati.es
Web de la revista: www.ati.es/reicis
E-mail: calidadsoft@ati.es
ISSN: 1885-4486

Publicado por la Asociación de Técnicos de Informática (ATI), Via Laietana, 46, 08003 Barcelona.

Secretaría de dirección: ATI Madrid, C/Padilla 66, 3º dcha., 28006 Madrid

Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS)

Contenidos

REICIS

Aplicación de un oráculo de prueba automatizado a la evaluación de salidas de programas basados en XML

Dae S. Kim-Park, Claudio de la Riva y Javier Tuya
Universidad de Oviedo
kim_park@lsi.uniovi.es, claudio@uniovi.es, tuya@uniovi.es

Resumen

La prueba de programas que procesan datos XML plantea diversos retos, entre los cuales destaca la obtención de un oráculo de prueba para dar soporte a la evaluación de las ejecuciones de pruebas. Para abordar este problema, en este trabajo se propone un oráculo de prueba automatizado dirigido a la prueba de programas de procesamiento de XML. El oráculo propuesto opera con una especificación del programa bajo prueba combinando dos niveles de especificación: (1) una de los requisitos de comportamiento particulares del programa bajo prueba, proporcionada por el ingeniero de pruebas, y (2) una especificación invariante del mecanismo de evaluación del oráculo, que determina si el programa cumple los requisitos de comportamiento suministrados. La automatización del oráculo está determinada por el uso de un lenguaje de especificación ejecutable, con el que se representan ambos niveles de especificación como código ejecutable. Se ilustra la aplicabilidad de oráculo mediante un caso de estudio que muestra resultados satisfactorios.

Palabras clave: prueba de software, automatización de pruebas, oráculos de prueba, programas de procesamiento XML.

Application of an automated test oracle to the evaluation of outputs from XML-based programs

Abstract

Testing of XML processing programs poses diverse challenges: obtaining a test oracle to assist the evaluation of the test executions is one of the most difficult. This work presents an automated test oracle for testing XML processing programs in order to address this problem. The proposed oracle operates with a specification of the program under test combining two specification levels: (1) one for the behavioural requirements of the program provided by the test engineer, and (2) an invariant specification of the evaluation mechanism of the oracle, intended to determine whether the program satisfies the given behavioural requirements. The oracle automation is determined by the use of an executable specification language which represents specification levels as executable code. The applicability of the oracle is illustrated through a case study that shows successful results.

Key words: software testing, test automation, test oracles, XML processing programs.

Kim-Park, D.S., de la Riva, C. y Tuya, J., "Aplicación de un oráculo de prueba automatizado a la evaluación de salidas de programas basados en XML", REICIS, vol. 7, no.1, 2011, pp.6-22. Recibido: 15-6-2010; revisado: 12-7-2010; aceptado:7-3-2011

1. Introducción y motivación

En la actualidad, el estándar XML [1] es el formato de intercambio de datos predominante en entornos distribuidos como la Web, en donde dispositivos heterogéneos de diferente hardware y sistema operativo interaccionan entre sí mediante el intercambio de información.

Habitualmente, los sistemas que forman parte de estos entornos disponen de componentes software especializados dedicados a efectuar las operaciones de acceso y manipulación sobre datos XML. Existen múltiples tecnologías con las que implementar estas operaciones, tales como los estándares XPath, XQuery, XSLT, SAX o DOM. Algunos de ellos son adecuados para operar sobre ficheros de datos XML, otros están adaptados a la consulta de recursos XML almacenados en repositorios de datos, o son capaces de manipular datos XML desde diferentes niveles de abstracción. Sin embargo, cualquier conjunto de operaciones de acceso y manipulación de datos XML, de forma independiente a su tecnología de implementación, puede considerarse en términos generales como un programa de procesamiento XML, entendiendo como tal cualquier artefacto software que toma como entrada un conjunto de datos XML, y realiza un procesamiento sobre los mismos produciendo nuevos datos XML en su salida.

La complejidad que los programas de procesamiento XML pueden alcanzar motiva la necesidad de someterlos a actividades de verificación y validación, y en particular a la prueba de software, la cual consiste en someter los artefactos software desarrollados a ejecuciones controladas con el objetivo de comprobar su funcionamiento. No obstante, la prueba del software lleva asociada diversos retos en el ámbito particular de los programas de procesamiento XML, puesto que los datos XML son compatibles con el modelo de datos semi-estructurado [2] y permiten diversas representaciones válidas de la misma información, pueden integrar datos de diferente grado de estructuración, y no necesiten estar restringidos por un esquema de datos fijo. Ello dificulta la obtención de casos de prueba para los programas de procesamiento XML, tanto en lo que respecta a la obtención de las entradas de las pruebas, como en lo relacionado a la evaluación de salidas de la prueba mediante un oráculo de prueba [3], entendiendo por oráculo de prueba cualquier mecanismo encargado de comprobar las salidas, y emitir en consecuencia un veredicto sobre la ejecución de la prueba que permita valorar si el programa se comporta de forma

correcta. Si el programa bajo prueba produce una salida inesperada a causa de un defecto, el oráculo debe ser capaz de detectar esta situación y notificarla.

En cuanto a la obtención de entradas de prueba para programas de procesamiento XML, existen algunas técnicas y herramientas que pueden emplearse para dar soporte a la generación automática de datos [4][5][6]. Por otro lado, en lo relativo a la obtención oráculos de prueba se aprecia una menor actividad investigadora, dado que el cuerpo de conocimiento en torno a los oráculos de prueba es generalmente escaso. En la actualidad no se dispone de oráculos específicos para programas de procesamiento XML y, en consecuencia, la evaluación de las salidas de las pruebas es realizada manualmente o con medios de escasa automatización, lo cual convierte la prueba de estos programas en una labor tediosa, propensa a errores y de alto coste debido al volumen y la complejidad de los datos que en muchas ocasiones es necesario evaluar. Es por ello necesario disponer de oráculos de prueba que faciliten la evaluación de los resultados de las pruebas en este tipo de programas.

Con estas consideraciones, este trabajo está enfocado a la definición, obtención y evaluación de un oráculo de prueba destinado a dar soporte a la prueba de programas de procesamiento XML requiriendo la mínima intervención humana posible. El oráculo propuesto basa su funcionamiento en dos niveles de especificación: una particular, proporcionada manualmente por el ingeniero de pruebas, con la cual se define el comportamiento esperado del programa bajo prueba mediante una notación simplificada, y otra general, integrada en el oráculo, definiendo un procedimiento invariante para comprobar las salidas y determinar los veredictos de la prueba. Codificando estos niveles de especificación con un lenguaje de especificación ejecutable, es posible alcanzar la automatización del oráculo. En las secciones siguientes se detallan estos aspectos y se lleva a cabo una evaluación de la efectividad de este oráculo a través de un caso de estudio.

El trabajo está estructurado de la forma siguiente. En la Sección 2 se describe el oráculo de prueba propuesto y su modo de operación dentro del escenario de prueba de los programas de procesamiento XML. En la Sección 3 se detalla la especificación ejecutable en dos niveles utilizada por el oráculo. En la Sección 4 se presenta el caso de estudio. Finalmente, la Sección 5 presenta las conclusiones y plantea líneas de trabajo futuro.

2. Descripción del oráculo de prueba

El oráculo propuesto opera sobre los datos de prueba empleando dos niveles de especificación: los *requisitos de comportamiento* y las *restricciones del oráculo*. Los *requisitos de comportamiento* describen características esperadas de un programa bajo prueba particular, mientras que las *restricciones del oráculo* son condiciones que describen las relaciones esperadas entre el comportamiento mostrado por el programa bajo prueba y los requisitos de comportamiento suministrados. Son por tanto condiciones necesarias para el correcto funcionamiento del programa bajo prueba, de modo que si son violadas revelan la presencia de defectos en el programa bajo prueba.

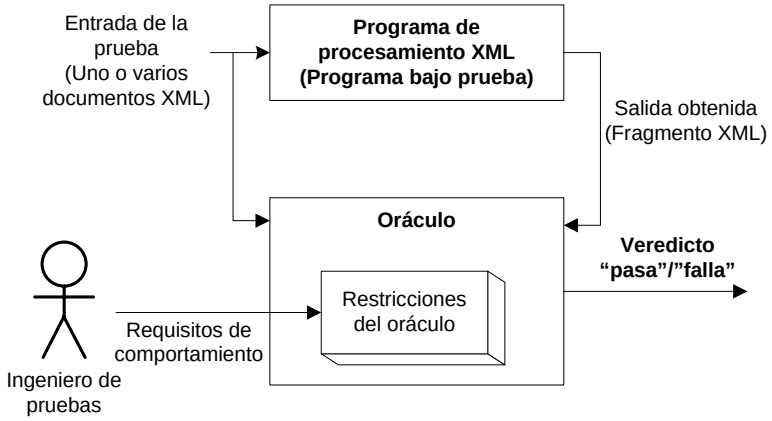

Figura 1. Escenario de prueba para programas de procesamiento XML

En la Figura 1 se muestra el escenario de prueba soportado por el oráculo. Cada programa de procesamiento requiere al menos una especificación de requisitos de comportamiento que describen el comportamiento esperado del programa bajo prueba. Estos requisitos de comportamiento son suministrados por el ingeniero de pruebas mediante la abstracción de la especificación esperada del programa, es decir, considerando un subconjunto menos restrictivo de la especificación. De este modo el coste de especificación manual se ve reducido. Por otro lado, las restricciones del oráculo, integradas en el mecanismo de evaluación del oráculo, no necesitan ser especificadas o conocidas por el ingeniero de pruebas, ya que son invariantes e independientes del programa sometido a la prueba.

El procedimiento para obtener los veredictos de las pruebas es totalmente automático una vez suministrados los requisitos de comportamiento. Durante la prueba, ante cada ejecución del programa, el oráculo efectuará la evaluación de las restricciones del oráculo permitiendo determinar si los datos de la prueba satisfacen los requisitos de

comportamiento. En caso de que los datos de la prueba violen alguna restricción con los requisitos de comportamiento suministrados, el oráculo emitirá un veredicto de fallo (veredicto "falla") indicando el motivo de la violación. En caso contrario, si todas las restricciones se satisfacen, el oráculo indicará que el programa ha pasado la prueba (veredicto "pasa"), en cuya situación el oráculo no revelará la presencia de defectos en el programa.

3. Especificación ejecutable del oráculo de prueba

Para automatizar el procedimiento de evaluación de salidas, se propone codificar los requisitos de comportamiento y restricciones mediante el lenguaje XQuery [7]. El propósito es utilizar XQuery como lenguaje de especificación ejecutable, cumpliendo la doble función de representar la especificación de las operaciones sobre datos XML con alto nivel de abstracción, y de poder ser ejecutado sobre instancias de datos de prueba particulares para derivar información útil desde la perspectiva del oráculo. El lenguaje XQuery es adecuado para este propósito ya que está especialmente concebido para implementar operaciones de acceso y manipulación de datos XML, y presenta una sintaxis más simplificada y de más alto nivel que otras alternativas existentes orientadas a operar sobre datos XML, como podrían ser las interfaces DOM o SAX.

En las secciones siguientes se describen los requisitos de comportamiento y las restricciones aceptadas por el oráculo. En un trabajo previo [8] se puede encontrar una definición formal de estos elementos de especificación.

3.1. Requisitos de comportamiento

Los requisitos de comportamiento pueden ser suministrados con el nivel de detalle que sea adecuado de acuerdo a los objetivos de la prueba, siendo posible omitir aquellos requisitos que no se consideren necesarios o que planteen dificultades en su identificación. En cualquier caso, de forma independiente al detalle de los requisitos, el oráculo será capaz emitir veredictos de fallo fiables, si bien la detección de los fallos será más imprecisa cuanto menor detalle tengan los requisitos de comportamiento.

De aquí en adelante en esta sección, se propone un catálogo de requisitos de comportamiento descritos en una notación basada en XQuery. Estos requisitos han de ser suministrados al oráculo en forma de script XQuery, a modo de fichero de configuración.

1) Implementación relajada del programa (`gs`). Es un programa de procesamiento de XML que, recibiendo cualquier entrada de prueba, produce un superconjunto de la salida esperada. Esto es, una salida que contiene los nodos y jerarquías XML de la salida esperada. Se obtiene implementando una versión relajada de la especificación esperada del programa bajo prueba. En particular, la relajación consiste en alterar la especificación sobre las operaciones de filtrado de datos, omitiendo operandos u operadores o sustituyendo a los mismos por otros de más fácil definición. Dado que la implementación de una versión relajada de la especificación implica implementar una especificación más simple, será en general un proceso menos costoso y menos susceptible a errores del que supondría implementar el programa bajo prueba. Las salidas relajadas obtenidas con esta implementación pueden ser empleadas como fuente de datos para determinar la corrección de las salidas del programa bajo prueba.

La implementación relajada se suministra encapsulada en una función XQuery con el siguiente prototipo.

```
declare function gs($I as item()*) as item()* {
      ... implementación relajada del programa bajo prueba ...
};
```

El parámetro $I representa una secuencia de una o más estructuras XML correspondientes a la entrada de la prueba. La función debe devolver la salida relajada, es decir, la salida producida por la implementación relajada.

2. Conjunto de nodos esperados (`$Es`). Es una variable de XQuery que permite enumerar una secuencia de nombres y tipos de nodos XML (elementos, atributos o nodos de texto [9]) que se esperan encontrar en la salida de toda ejecución del programa bajo prueba. Se suministra como una variable XQuery de acuerdo al esquema siguiente.

```
declare variable $Es :=
<expectedNodes>
      <node type="tipo de nodo" data="nombre o valor del nodo"/>
      ...
      <node type="tipo de nodo" data="nombre o valor del nodo"/>
</expectedNodes>;
```

Cada nodo especificado puede ser de tipo elemento, atributo o texto (`element`, `attribute`, `text`), y los datos pueden definir o bien el nombre del nodo si se trata de un elemento o atributo, o bien su valor si se trata de un nodo de texto.

2) Conjunto de nodos inesperados (`$Us`). Similar al conjunto de nodos esperados, permite enumerar los nodos que no han de estar presentes en ninguna salida del programa bajo prueba. Su declaración en XQuery sigue el esquema siguiente, con el mismo convenio que el definido para el conjunto de nodos esperados (`$Es`).

```
declare variable $Us :=
<unexpectedNodes>
      <node type="tipo de nodo" data="nombre o valor del nodo"/>
      ...
      <node type="tipo de nodo" data="nombre o valor del nodo"/>
</unexpectedNodes>;
```

3) Conjunto de ordenación esperada (`$Rs`). Es una variable de XQuery que permite especificar la ordenación lexicográfica esperada de determinados nodos de la salida. Sigue el esquema de declaración mostrado a continuación.

```
declare variable $Rs :=
<ordering>
      <order nodes="ruta hasta nodo" type="tipo de ordenación"/>
      ...
      <order nodes="ruta hasta nodo" type="tipo de ordenación"/>
</ordering>;
```

La *ruta hasta el nodo* es una expresión XPath [7] que determina el conjunto de nodos al que se refiere el requisito de orden, mientras que el *tipo de ordenación* puede ser ascendente (`ascending`) o descendente (`descending`).

4) Conjunto de cardinalidad esperada (`$Cs`). Es una variable de XQuery que permite especificar el número de nodos descendientes que debe haber bajo ciertos nodos de la salida. Su esquema de declaración es el siguiente.

```
declare variable $Cs :=
<cardinalities>
    <cardinality nodes="ruta hasta nodo"
                 contents="contenidos del nodo"
                 minInclusive="cardinalidad mínima"
                 maxInclusive="cardinalidad máxima"/>
          ...
</cardinalities>;
```

Cada requisito de cardinalidad especifica la ruta XPath de un conjunto de nodos objetivo, sus nodos descendientes (contenidos) a considerar y las cardinalidades mínima y máxima que deben satisfacer.

En la Sección 4 se presenta un caso de estudio donde se ejemplifica una instancia concreta de estos requisitos.

3.2. Restricciones del oráculo

Las restricciones del oráculo son condiciones necesarias para la corrección del programa bajo prueba, destinadas a determinar si los datos de prueba satisfacen los requisitos de comportamiento. Estas restricciones forman parte del procedimiento de comprobación del oráculo y no requieren ser manipuladas por el ingeniero de pruebas para efectuar la prueba. Se plantean seis restricciones de acuerdo al catálogo de posibles requisitos propuesto (Sección 3.1).

Restricción 1: Inclusión en la salida relajada: Los nodos de la salida del programa bajo prueba deben estar contenidos en la salida producida por la implementación relajada (`gs`), pero no en el conjunto de nodos inesperados (`$Us`).

Restricción 2: Inclusión de nodos raíz en la salida relajada: Los nodos raíz de la salida de la prueba deben estar contenidos entre los nodos raíz de la salida producida por la implementación relajada (`gs`), pero no entre los nodos inesperados (`$Us`).

Restricción 3: Consistencia de la jerarquía: Para todo par de nodos que cumplan la relación padre-hijo en la salida esperada, debe existir otro par de nodos equivalente en la salida de la implementación relajada cumpliendo la misma relación.

Restricción 4: Presencia de nodos esperados: La salida del programa bajo prueba debe contener los nodos esperados (`$Es`).

Restricción 5: Ordenación correcta: Los nodos de la salida obtenida deben estar correctamente ordenados de acuerdo al conjunto de ordenación esperada (`$Rs`).

Restricción 6: Rangos de cardinalidad esperados: Los nodos de la salida obtenida deben cumplir los requisitos de cardinalidad descritos en el conjunto de cardinalidad esperada (`$Cs`).

Cada una de las restricciones ha sido implementada como una función XQuery que comprueba el cumplimiento de la condición establecida por la restricción. En caso de que la condición no se satisfaga, la función produce un mensaje basado en XML describiendo la causa de la violación. Como ejemplo, en la Figura 2 se muestra la implementación de la restricción 1, donde los parámetros $p_I, $gs_I y $Us representan la salida obtenida en la ejecución de la prueba, la salida de la implementación relajada y el conjunto de nodos

inesperados, respectivamente. Como puede observarse, cuando la restricción es violada, la función produce un mensaje de error representado por un nodo XML failure. El resto de restricciones sigue un patrón de implementación análogo.

```
declare function constraints:constraint1(
      $gs_I as item()*,
      $Us as item()*,
      $p_I as item()*)
as item()*
{
  let $gsI_Us := nodes:delete-from-nodeset($gs_I, $Us)
  for $n in nodes:distinct-nodes($p_I)
  return
    if(exists(nodes:equivalent-nodes($gsI_Us, $n))
    then
      ()
    else
      <failure constraint="1"
       message="Unexpected node found in the test
                output: {nodes:print($n)}"/>
};
```

Figura 2. Implementación de la restricción 1 (inclusión en la salida relajada).

Las restricciones están integradas en el procedimiento del oráculo, el cual implementa la ejecución de la función XQuery de cada restricción, la captura de los mensajes de fallo producidos por cada función, y la emisión del veredicto, "pasa" si no se han producido mensajes de fallo, o "falla" en otro caso. A su vez, este procedimiento debe integrarse en un arnés de pruebas para automatizar la captura de los datos de prueba, aunque los detalles sobre el mismo están fuera del ámbito de este trabajo.

4. Caso de estudio

En este caso de estudio se ilustra la aplicación del oráculo automatizado para un supuesto programa bajo prueba. Se asume que se dispone de las restricciones del oráculo representadas en forma de especificación XQuery (Sección 3.2), y que éstas están integradas en un arnés de pruebas apropiadamente como requiere el entorno de prueba. Por tanto, se considera que el oráculo automatizado es obtenido una vez que el ingeniero de pruebas suministra los requisitos de comportamiento (Sección 3.1) del programa de estudio. Por otro lado, el caso de estudio tiene como fin evaluar la efectividad del oráculo de prueba

así obtenido, para lo cual se emplea una técnica basada en perturbación de datos. Los pasos seguidos en este caso de estudio son los siguientes.

1. Se selecciona un programa de procesamiento XML objetivo (Sección 4.1).
2. Obtención del oráculo automatizado: Se suministra al oráculo un conjunto de requisitos de comportamiento que especifican el comportamiento esperado del programa seleccionado en el paso 1 (Sección 4.2).
3. Evaluación de la efectividad del oráculo.
    3.1. Se toma un caso de prueba de referencia para el programa seleccionado que incluye una entrada y una salida esperada. Aplicando la técnica de perturbación de datos [10] sobre la salida esperada, se obtienen múltiples instancias de salidas con fallos, cada una de las cuales simula un resultado incorrecto del programa (Sección 4.3).
    3.2. Se ejecuta el oráculo con los requisitos de comportamiento suministrados en el punto 2 y los datos de prueba con fallos simulados obtenidos en el paso 3, se capturan los veredictos producidos por el oráculo (Sección 4.4) y se analizan los resultados.

La efectividad del oráculo se estima en función del número de salidas con fallos que el oráculo juzga satisfactoriamente con un veredicto de fallo. Este método es adecuado, en tanto en cuanto el mecanismo de comprobación del oráculo se basa en restricciones que definen condiciones necesarias para la corrección del programa. Como consecuencia de ello, cada vez que el oráculo emite un veredicto de fallo—y por tanto se viola alguna restricción—, se puede afirmar que dicho veredicto es fiable y está determinado por las capacidades del oráculo. En cambio, no sería factible evaluar la efectividad del oráculo en función de los veredictos positivos (veredicto "pasa"), ya que si no se produce la violación de ninguna restricción, el veredicto no es concluyente y no indica una medida de efectividad admisible.

En las secciones siguientes se detallan las acciones y resultados intermedios de cada uno de los pasos seguidos durante el experimento, y finalmente se discuten los resultados obtenidos.

Entrada del programa (*bib.xml*):

```
<bib>
    <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
    <book year="2000">
        <title>Data on the Web</title>
        <author><last>Abiteboul</last><first>Serge</first></author>
        <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
    </book>
    <book year="1999">
        <title>The Economics of Technology and Content for Digital TV</title>
        <editor>
               <last>Gerbarg</last><first>Darcy</first>
                <affiliation>CITI</affiliation>
        </editor>
            <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
    </book>
</bib>
```

Salida esperada:

```
<bib>
    <book-pair>
        <title>TCP/IP Illustrated</title>
        <title>Advanced Programming in the Unix environment</title>
    </book-pair>
</bib>
```

Figura 3. Ejemplo de ejecución del programa del caso de estudio.

4.1. Selección del programa objetivo

Como programa objetivo para el estudio se ha seleccionado el programa de procesamiento XML identificado como *Q12* del Caso de Uso *XMP* del W3C [11]. El programa seleccionado tiene el cometido de procesar un fichero (*bib.xml*) que almacena información bibliográfica en formato XML, entre la que se incluye el título, los autores o editores, la fecha de publicación, la editorial y el precio de una serie de libros. Debe procesar este

contenido y producir los pares de títulos diferentes de aquellos libros que tengan los mismos autores. La Figura 3 muestra un ejemplo de ejecución con la entrada del programa y la salida esperada.

```
declare function gs($input as item()*) as item()*
{
    <bib> {
        for $t1 in $input//title
        for $t2 in $input//title
        where $t1 ne $t2
        return
            element book-pair {$t1, $t2}
    }</bib>
};


declare variable $Cs :=
   <cardinalities>
      <cardinality nodes="/bib/book-pair"
                   contents="/*"
                   minInclusive="2"
                   maxInclusive="2"/>
   </cardinalities>
```

Figura 4. Requisitos de comportamiento para el programa bajo prueba

4.2. Especificación de requisitos de comportamiento

Se ha especificando el comportamiento esperado del programa seleccionado (Sección 4.2) suministrando los requisitos de comportamiento mostrados en la Figura 4. Este conjunto de requisitos incluye:

- Una implementación relajada (función XQuery `gs`), obtenida relajando la condición sobre los autores de los libros, de modo que la implementación produce todos los posibles pares de títulos diferentes que se proporcionen como entrada, sin importar cuáles sean sus autores.
- Un conjunto de cardinalidad esperada (variable XQuery `$Cs`) indicando que cada par de libros devuelto (`book-pair`) en la salida incluye dos elementos—cardinalidad en el rango [2, 2]—correspondientes al título de cada libro.

El resto de requisitos de comportamiento no ha sido definido en este caso particular. La intención es mostrar que es posible suministrar requisitos de comportamiento con bajo coste de especificación, y aunque ello pueda incurrir en cierta pérdida de precisión del oráculo, la efectividad final puede ser razonablemente alta, como se muestra más adelante.

4.3. Generación de salidas con fallos simulados

Las salidas con fallos simulados para evaluar el oráculo se han obtenido de forma automática mediante la técnica de perturbación de datos [10]. Esta técnica consiste en aplicar operadores de perturbación (alteración) sobre determinados datos de referencia para generar nuevas instancias de datos útiles para la prueba.

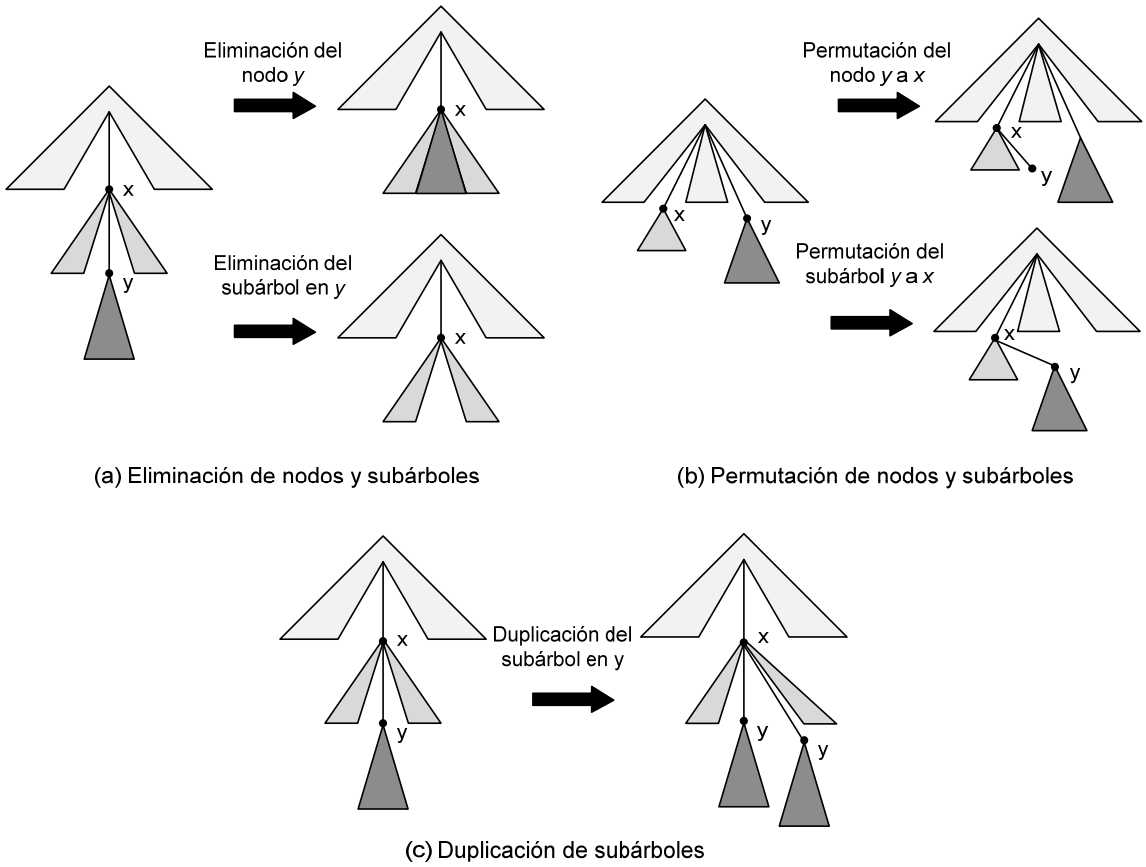

Figura 5. Efecto de las operaciones de perturbación sobre estructuras en forma de árbol (XML).

En este caso de estudio, los operadores de perturbación han sido aplicados sobre una salida esperada (correcta) de un caso de prueba de referencia, dando como resultado las diversas instancias de salidas con fallos. Concretamente, como caso de prueba se ha empleado el par entrada/salida de la Figura 3. Los operadores de perturbación de datos empleados se describen a continuación.

1. **Eliminación de nodos y subárboles** (Figura 5a). La eliminación de un nodo XML consiste en eliminar un elemento, atributo, o un nodo de texto, adjuntando, si los tuviere, sus nodos hijo al nodo ancestro inmediato. Por otro lado, la

eliminación de un subárbol consiste en eliminar un nodo y todos sus descendientes.

2. **Permutación de nodos y subárboles** (Figura 5b). Consiste en cambiar la posición de un nodo o un subárbol dentro de la jerarquía XML.
3. **Duplicación de subárboles** (Figura 5c). Consiste en duplicar un subárbol e insertar el duplicado como hermano inmediato del subárbol original.

Aplicando estos operadores de perturbación sobre cada nodo y subárbol de la salida esperada de referencia, se han obtenido en total 122 ejemplares de salidas fallidas, cada una incluyendo una perturbación diferente. No se contabilizan ejemplares que han sido descartados por no representar estructuras XML válidas, tales como salidas vacías o atributos XML sin nodo padre. Por tanto, los ejemplares no descartados son sintácticamente correctos aunque representan salidas que incumplen la especificación del programa. Con esto se pretende que la evaluación de la efectividad del oráculo se enfoque a los aspectos funcionales del programa, y no a la sintaxis de XML.

4.4. Resultados obtenidos

Tras ejecutar el oráculo con los requisitos de comportamiento especificados (Sección 4.2) y los datos de prueba simulados mediante perturbación (Sección 4.3), se han obtenido los resultados de la Tabla 1.

Operador	Nº salidas con fallo (*n)*	Nº salidas con fallo detectado (*m)*	Efectividad (%) *(100 · m/n)*
Eliminación de nodos	6	4	66,67
Eliminación de subárboles	5	2	40,00
Permutación de nodos	71	71	100,00
Permutación de subárboles	34	34	100,00
Duplicación de subárboles	6	4	66,67

Tabla 1. Resultados de la ejecución del oráculo sobre las salidas con fallos

La primera columna de la tabla indica el operador de perturbación que se ha empleado sobre la salida esperada de referencia. La segunda columna representa el número de instancias de salidas con fallos que se han derivado al aplicar el operador de perturbación. La tercera columna indica el número de salidas que el oráculo ha juzgado

correctamente con un veredicto de fallo. Por último, la cuarta columna indica el porcentaje de salidas juzgadas correctamente, lo cual da una idea de la efectividad del oráculo ante cada tipo de perturbación aplicada.

4.5. Discusión de los resultados

De los resultados anteriores, en general se aprecia que el oráculo es capaz de producir un alto número de veredictos correctos, aun con requisitos de comportamiento no muy precisos.

La efectividad del oráculo es máxima en la detección de fallos producidos por permutaciones inesperadas de nodos y subárboles. Estos fallos han sido detectados por las restricciones del oráculo 2, 3 y 6 (descritos en la Sección 3.2), encargadas de comprobar la correspondencia entre jerarquías XML de la salida con las salidas de la implementación relajada suministrada (gs).

Las perturbaciones por eliminación de nodos han sido detectadas satisfactoriamente por las restricciones del oráculo 3 y 6, debido a las inconsistencias de jerarquías que produce la eliminación de nodos. El oráculo muestra ciertas dificultades para detectar los fallos debidos a la eliminación de subárboles, ya que para detectar este tipo de perturbaciones, el oráculo sólo ha dispuesto del requisito de cardinalidad, soportado por la restricción 6.

Por último, las duplicaciones de subárboles son detectadas, o bien con la restricción del oráculo 6, o bien con la restricción 1 cuando la duplicación se produce sobre nodos de texto, caso en el cual la restricción detecta la presencia de nodos de texto inesperados (la duplicación de subárboles aplicada sobre un nodo de texto, da lugar dos nodos de texto con igual valor, lo que es interpretado por el oráculo como un único nodo de texto cuyo valor es la concatenación de los valores de ambos nodos).

Queda como incógnita valorar si la relajación de requisitos llevada a cabo en este estudio es realista desde un punto de vista práctico, ya que en este caso particular el ingeniero de pruebas podría suministrar requisitos de comportamiento más detallados sin que ello le supusiera un incremento notable en el coste de especificación. En cualquier caso, un una especificación más detallada daría lugar a un oráculo de mayor efectividad.

En [8] se ha hecho una evaluación más exhaustiva con un mayor número de programas e instancias de datos perturbados que ha llevado a las mismas conclusiones.

5. Conclusiones y trabajo futuro

Se ha presentado un oráculo automatizado para dar soporte a la prueba de programas de procesamiento XML. El oráculo opera con requisitos simplificados del comportamiento del programa bajo prueba suministrados por el ingeniero de pruebas, y restricciones que definen el procedimiento de evaluación de las salidas de las pruebas. Los requisitos de comportamiento y las restricciones del oráculo han sido implementados con un lenguaje de especificación ejecutable facilitando así la automatización del oráculo.

Se ha evaluado la aplicabilidad del oráculo a través de un caso de estudio en donde se muestra que el oráculo que es capaz de detectar un número de fallos razonable, aun cuando los requisitos de comportamiento del programa se suministran con baja precisión y, por tanto, bajo coste.

Para mejorar las capacidades de detección de fallos del oráculo, se plantea como trabajo futuro identificar y estudiar requisitos de comportamiento y restricciones del oráculo adicionales, tratando de mantener un bajo coste de especificación manual de cara a la aplicación del oráculo en un entorno de pruebas real. También se considerará la posibilidad de mejorar la especificación del oráculo con información derivada del criterio de selección empleado para obtener las entradas de prueba, ya que mucha información derivada del diseño de los datos de entrada podría permitir caracterizar el tipo de fallos se esperan encontrar en la salida de prueba.

6. Agradecimientos

Este trabajo ha sido financiado por el Gobierno del Principado de Asturias con la beca PCTI-FICYT (Ref. BP09080), y ha sido parcialmente financiado por el Ministerio de Educación y Ciencia de España dentro del Plan Nacional I+D+i, a través del proyecto Test4DBS (TIN2010-20057-C03-01).

Referencias

[1] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, 28 de noviembre de 2008. http://www.w3.org/TR/xml/ (último acceso: 2010).

[2] Abiteboul, S., "Querying Semi-Structured Data". En: Afrati, F. N., Kolaitis, P. G. (eds.), *Proceedings of the 6th International Conference on Database Theory. Delfos (Grecia)*, *8-10 de enero de 1997*, pp. 1-18, 1997.

[3] Weyuker, E. J., "On Testing Non-testable Programs", *The Computer Journal*, vol. 25, nº 4, pp. 465-470, 1982.

[4] Barbosa, D., Mendelzon, A. , Keenleyside, J. y Lyons, K., "ToXgene: a template-based data generator for XML". En: Franklin, M. J., Moon, B. y Ailamaki, A. (eds.), *Proceedings of the 2002 SIGMOD International Conference on Management of Data*. *Madison, Wisconsin (EEUU)*, *3-6 de junio de 2002*, pp. 616-616, 2002.

[5] Bertolino, A., Gao, J., Marchetti, E., Polini, A., "TAXI—A tool for XML-Based Testing". En: Kawada, S. (ed.), *Proceedings of the 29th International Conference on Software Engineering*. *Minneapolis, Minnesota (EEUU)*, *20-26 de mayo de 2007*, pp. 53-54, 2007.

[6] de la Riva, C., García-Fanjul, J. y Tuya, J., "A Partition-Based Approach for XPath Testing", En: IARIA Logistics, IEEE CS Press (eds.), *Proceedings of the International Conference on Software Engineering Advances*. *Tahití (Polinesia Francesa)*, *29 de octubre-3 de noviembre de 2006*, 2006, pp. 17-22, 2006.

[7] World Wide Web Consortium, *XQuery 1.0: An XML Query Language*, 23 de enero de 2007. http://www.w3.org/TR/xquery/ (último acceso: 2010).

[8] Kim-Park, D. S., de la Riva, C. y Tuya, J., "An Automated Test Oracle for XML Processing Programs". En: ACM, *Proceedings of the 1st International Workshop on Software Test Output Validation*. *Trento (Italia)*, *13 de julio de 2010*, pp. 5-12, 2010.

[9] World Wide Web Consortium, *XQuery 1.0 and XPath 2.0 Data Model (XDM)*, 23 de enero de 2007. http://www.w3.org/TR/xpath-datamodel/ (último acceso: 2010).

[10] Xu, W., Offut, J., y Luo J., "Testing Web Services by XML Perturbation". En: Kawada, S. (ed.), *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*. *Chicago, Illinois (EEUU)*, 8-11 de noviembre de 2005, pp. 257-266, 2005.

[11] World Wide Web Consortium, *XML Query Use Cases*, 23 de marzo de 2007, http://www.w3.org/TR/xquery-use-cases/ (último acceso: 2010).