

Oráculo de prueba automatizado para programas de procesamiento XML

Dae S. Kim-Park, Claudio de la Riva, Javier Tuya

Departamento de Informática

Campus de Viesques

Universidad de Oviedo

33204 Gijón

kim_park@lsi.uniovi.es, claudio@uniovi.es, tuya@uniovi.es

Resumen

Se presenta un oráculo de prueba automatizado para dar soporte a la prueba de programas destinados al procesamiento de datos XML. La automatización del oráculo está determinada principalmente por el uso de un lenguaje de especificación ejecutable con el cual se representa la especificación de los programas bajo prueba combinando dos niveles de especificación: (1) una especificación de los requisitos de comportamiento particulares del programa bajo prueba, proporcionada de forma manual de acuerdo a los objetivos de la prueba, y (2) una especificación formando parte del mecanismo de evaluación del oráculo, compuesta por un conjunto de restricciones que describen el comportamiento común a todos los programas de procesamiento XML. Se ilustra la aplicación de oráculo a través de un caso de estudio que muestra resultados satisfactorios.

1. Introducción y motivación

El estándar XML [7] es actualmente el formato de datos universal que predomina en entornos como la Web, en donde dispositivos heterogéneos con diferente hardware y sistemas operativos 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 de datos XML requeridas. Existen múltiples tecnologías con las que es posible implementar estas operaciones, como XPath, XQuery, XSLT, SAX o

DOM. Algunas de ellas son adecuadas para operar sobre ficheros con datos XML, otras están adaptadas a la consulta de recursos XML almacenados en repositorios de datos, o son capaces de manipular datos XML en diferentes niveles de detalle. Sin embargo, a pesar de sus diferencias tecnológicas, cualquier conjunto de operaciones de acceso y manipulación de datos XML puede considerarse en términos generales como un *programa de procesamiento XML*, entendiéndose como tal cualquier artefacto software de caja negra que toma como entrada un conjunto de datos XML, y produce nuevos datos XML.

Los programas de procesamiento XML pueden implementar operaciones de muy diversa complejidad para procesar entradas XML y calcular y construir sus salidas, por lo que resulta importante verificar y validar su funcionamiento sometiénolos a pruebas. No obstante, la prueba de este tipo de programas plantea una serie de retos derivados de las características propias de los datos XML que manejan.

Los datos XML presentan las características del modelo de datos semi-estructurado [1]: permiten que un mismo tipo de dato tenga diversas representaciones válidas, pueden definir datos con diferentes tipos de estructuración (desde datos muy estructurados hasta datos completamente desestructurados), y sus datos no necesitan estar restringidos por un esquema. Todo ello dificulta la obtención de casos de prueba para programas de procesamiento XML, tanto en lo que respecta a la obtención de las entradas de las pruebas, como en lo relacionado con la obtención de un *oráculo de prueba* [6], conocido como cualquier principio o mecanismo encargado de comprobar las salidas de la prueba con el objetivo

de emitir un veredicto acerca de la corrección del programa.

En cuanto a la obtención de entradas de prueba, existen varias técnicas y herramientas que pueden emplearse para dar soporte a la generación automática de datos [2][3][5]. Sin embargo, en cuanto a la obtención oráculos de prueba se aprecia una menor actividad investigadora, y 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 la evaluación de las salidas de las pruebas se realiza manualmente, lo cual tiende a ser una labor tediosa, propensa a errores y de alto coste debido al volumen y complejidad que pueden alcanzar los datos XML de salida.

Con estas consideraciones, en este trabajo se presenta un oráculo de prueba para dar soporte a la prueba de programas de procesamiento XML. El oráculo presentado basa su procedimiento de evaluación de salidas en dos niveles de especificación, una particular, proporcionada manualmente por el tester, que describe el comportamiento esperado del programa bajo prueba, y otra general integrada en el mecanismo del oráculo de prueba que define el procedimiento para determinar los veredictos de las pruebas. El oráculo ha sido automatizado codificando estos niveles de especificación con un lenguaje de especificación ejecutable.

Este trabajo se estructura de la forma siguiente. En la Sección 2 se describe el oráculo de prueba 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 aceptada por el oráculo. En la Sección 4 se presenta un caso de estudio con un programa bajo prueba de ejemplo. Finalmente, la Sección 5 presenta las conclusiones y plantea las líneas de trabajo futuro.

2. Descripción del oráculo de prueba

El oráculo opera sobre las salidas de la prueba con dos niveles de especificación: los requisitos de comportamiento y las restricciones del oráculo. Los *requisitos de comportamiento* son atributos que describen el comportamiento de un programa bajo prueba particular. Las *restricciones del oráculo* son condiciones que, dependientes de los

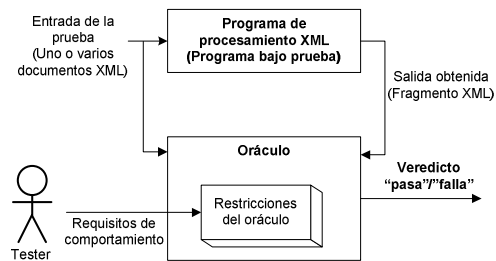


Figura 1. Escenario de prueba de programas de procesamiento XML

requisitos de comportamiento, describen comportamientos generales que todo programa de procesamiento XML correctamente implementado debe cumplir. Son por tanto condiciones necesarias que al ser violadas revelan la presencia de defectos en el programa bajo prueba.

En la Figura 1 se muestra el escenario de prueba soportado por el oráculo. Durante la prueba, cada programa de procesamiento requiere al menos una especificación de requisitos de comportamiento suministrada manualmente por el *tester*. Por otro lado, las restricciones del oráculo son independientes de los detalles del programa bajo prueba, por lo que se encuentran integradas en el mecanismo de evaluación del oráculo y el tester no necesita especificarlas o conocerlas.

El procedimiento de evaluación de salidas se lleva a cabo a través de un proceso automático en el cual se parametriza cada restricción del oráculo con los requisitos de comportamiento del programa y los datos de ejecución de la prueba, constituidos por la entrada, formada por uno o varios documentos XML [7], y la salida obtenida en forma de fragmento XML—cualquier estructura XML sintácticamente correcta—. A continuación, el oráculo comprueba el cumplimiento de cada una de las restricciones. Si alguna de ellas no se satisface, el oráculo emite un veredicto de fallo (veredicto “falla”) indicando el motivo de la violación. En caso contrario, si todas las restricciones se satisfacen, el oráculo indica que el programa ha pasado la prueba (veredicto “pasa”), en cuya situación el oráculo no revela la presencia de defectos en el programa.

3. Especificación ejecutable del oráculo

Para automatizar el procedimiento de evaluación de salidas, los requisitos de comportamiento y las restricciones del oráculo se han codificado manualmente mediante el lenguaje XQuery [10]. Este lenguaje 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 de programación DOM o SAX. Con ello, el lenguaje XQuery se ha empleado como lenguaje de especificación ejecutable, cumpliendo la doble función de permitir la especificación de operaciones sobre datos XML con un alto nivel de abstracción, y de ser ejecutado sobre instancias de datos particulares para producir ciertos resultados útiles para el oráculo.

A continuación se describen los requisitos de comportamiento y las restricciones aceptadas por el oráculo. En un trabajo previo [4] 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, pudiendo incluso omitirse algunos de ellos si no se consideran necesarios. En cualquier caso el oráculo podrá emitir veredictos de fallo fiables, aunque la detección de los fallos será más imprecisa cuanto menor detalle tengan los requisitos de comportamiento. De aquí en adelante en esta sección, se definen los requisitos de comportamiento disponibles para especificar el comportamiento del programa bajo prueba.

Implementación relajada del programa (gs). Es un programa de procesamiento de XML implementado en el lenguaje XQuery que, recibiendo cualquier entrada de prueba, produce un superconjunto de la salida esperada. El programa así implementado contendrá menos operaciones de filtrado de datos que el programa bajo prueba, por lo que se espera que su implementación sea más sencilla y menos propensa a errores que la del programa bajo prueba. La salida producida por esta implementación se considera una aproximación de la salida esperada.

Conjunto de nodos esperados (\$Es). Es una variable de XQuery que enumera una secuencia de nombres y tipos de nodos XML (elementos, atributos o nodos de texto [9]) que han de estar incluidos en la salida de todas las ejecuciones del programa bajo prueba.

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.

Conjunto de ordenación esperada (\$Rs). Es una variable de XQuery que especifica la ordenación esperada de los nodos de la salida (ordenación lexicográfica del valor de los nodos, ascendente o descendente).

Conjunto de cardinalidad esperada (\$Cs). Es una variable de XQuery que permite especificar el número de nodos hijo que debe haber bajo ciertos nodos de la salida.

3.2. Restricciones del oráculo

Las restricciones del oráculo son condiciones necesarias para la corrección del programa bajo prueba, destinadas a comprobar si los datos de ejecución de las pruebas satisfacen los requisitos de comportamiento. Estas restricciones están definidas formando parte del oráculo, y no necesitan ser suministradas ni modificadas por el tester para efectuar las pruebas.

Cada una de las restricciones ha sido implementada como una función XQuery que comprueba el cumplimiento de la condición establecida por la restricción. Cada función se encuentra parametrizada con los datos de la prueba y los requisitos de comportamiento y, en caso de que la restricción sea violada, produce un mensaje que describe el fallo desencadenante de la violación.

A continuación se describen las restricciones del oráculo indicando la condición que deben satisfacer para que el oráculo no emita el veredicto de fallo.

Restricción 1: Inclusión en la salida relajada: Los nodos producidos por el programa bajo prueba deben estar contenidos en la salida producida por la implementación relajada (gs) pero no en el conjunto de nodos inesperados (\$Us). Como ejemplo, en la Figura 2 se muestra la implementación de esta restricción. Como puede observarse, si la restricción es violada, se produce

un mensaje de error representado por un nodo

```

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.

failure. El resto de restricciones sigue un patrón de implementación análogo.

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 en la salida de la implementación relajada otro par de nodos equivalente que cumpla la misma relación.

Restricción 4: Presencia de nodos esperados: La salida del programa bajo prueba contiene los nodos esperados (**Es**).

Restricción 5: Ordenación correcta: Los nodos de la salida obtenida se encuentran correctamente ordenados de acuerdo al conjunto de ordenación esperada (**Rs**).

Restricción 6: Rangos de cardinalidad esperados: Los nodos de la salida obtenida cumplen los requisitos de cardinalidad impuestos por el conjunto de cardinalidad esperada (**Cs**).

4. Caso de estudio

En esta sección se ilustra la aplicabilidad del oráculo de prueba a través de un caso de estudio experimental que consta de los siguientes pasos:

1. Inicialmente, se selecciona un programa de procesamiento XML objetivo (Sección 4.1).

2. 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. 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 [11] sobre la salida esperada, se obtienen múltiples instancias de salidas con fallos, simulando resultados incorrectos de la ejecución del programa (Sección 4.3).

4. 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, y se recogen los resultados (Sección 4.4).

Tras completar estos pasos, se ha medido la efectividad del oráculo en función del número de salidas con fallos simulados que el oráculo ha juzgado con un veredicto de fallo. Este método es adecuado en tanto en cuanto el mecanismo de comprobación del oráculo se basa en restricciones que han sido definidas en forma de 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 ha violado 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 según los veredictos positivos (veredicto “pasa”) que emita ante ejecuciones correctas, ya que si no se produce la violación de ninguna restricción, el veredicto no es concluyente y no indica ninguna medida de efectividad fiable.

4.1. Selección del programa objetivo

Se ha seleccionado el programa mostrado en la Figura 3. Este programa ha sido extraído del Caso de Uso XMP del W3C [8] (identificado con el nombre Q12). Nótese que aunque el programa está implementado en el lenguaje XQuery, desde el punto de vista del oráculo podría estar implementado con cualquier otro lenguaje siempre y cuando con el enfoque de caja negra presente el comportamiento de un proceso con entradas y

salidas XML acordes al escenario de prueba de la Sección 2.

```

<bib>
{
  for $book1 in doc("bib.xml")//book,
  $book2 in doc("bib.xml")//book
  let $aut1 := for $a in $book1/author
              order by $a/last, $a/first
              return $a
  let $aut2 := for $a in $book2/author
              order by $a/last, $a/first
              return $a
  where $book1 << $book2
  and not($book1/title = $book2/title)
  and deep-equal($aut1, $aut2)
  return
    <book-pair>
      { $book1/title }
      { $book2/title }
    </book-pair>
}
</bib>

```

Figura 3. Programa bajo prueba de ejemplo

El programa seleccionado tiene como cometido acceder a un fichero (*bib.xml*) que almacena información bibliográfica en formato XML, y procesar su contenido para producir los pares de títulos diferentes de los libros que tengan los mismos autores.

4.2. Especificación de requisitos de comportamiento

Se ha especificando el comportamiento esperado del programa 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; es decir, la implementación relajada devuelve todos los posibles pares de títulos diferentes. Esta implementación es más sencilla y menos propensa a errores que el programa bajo prueba de la Figura 3, aunque en lugar de producir la salida esperada, produce un superconjunto de la misma (contiene la salida esperada).
- Un conjunto de cardinalidad esperada (variable XQuery *\$Cs*) indicando que cada par de libros devuelto está formado por 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

```

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

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.

4.3. Generación de salidas con fallos

Las salidas con fallos se han obtenido mediante la técnica de perturbación de datos [11]. Esta técnica consiste en aplicar operadores de perturbación (alteración) sobre determinados datos de referencia para generar nuevos datos útiles para la prueba. En este caso de estudio, los operadores de perturbación han sido aplicados sobre una salida esperada (correcta) de un caso de prueba, dando como resultado diversas instancias de salidas con fallos. Concretamente, como caso de prueba de referencia se ha empleado el par entrada/salida del programa Q12 proporcionado en el Caso de Uso XMP [8].

Los operadores de perturbación empleados (detallados con mayor profundidad en [4]) han sido: (1) la eliminación de nodos XML individuales y la eliminación de subárboles XML (nodos XML y sus descendientes) de la salida, (2) la permutación de nodos y subárboles—intercambiando nodos y subárboles entre sí dentro de la jerarquía XML—, y (3) la duplicación de subárboles XML.

Se han obtenido en total 122 ejemplares de salidas con fallos, cada una de las cuales ha

resultado de aplicar uno de los operadores sobre un nodo o subárbol de la salida esperada de referencia.

4.4. Resultados obtenidos

Los resultados obtenidos tras ejecutar el oráculo con los requisitos de comportamiento suministrados (Sección 4.2) y los datos de prueba simulados (Sección 4.3), se muestran en la Tabla 1. 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, el cual da una idea de la efectividad del oráculo ante cada tipo de perturbación.

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

| Operador | Nº salidas | Nº salidas con fallo detectado | Efectividad (%) |
|---------------------------|------------|--------------------------------|-----------------|
| 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 |

4.5. Discusión de los resultados

Los resultados muestran que aun sin disponer de requisitos de comportamiento muy precisos, el número de veredictos correctos producidos por el oráculo ha sido alto.

El oráculo muestra una efectividad máxima en la detección de fallos producidos por permutaciones inesperadas de nodos y subárboles, las cuales han sido detectadas por las restricciones del oráculo 2, 3 y 6 (ver Sección 3.2).

Las perturbaciones por eliminación de nodos han sido detectadas satisfactoriamente por las restricciones del oráculo 3 o 6, debido a que la eliminación de nodos produce inconsistencias de la jerarquía padre-hijo de los nodos. Existen mayores dificultades para detectar los fallos debidos a la eliminación de subárboles, ya que para detectar este tipo de perturbaciones, el oráculo sólo dispone de los requisitos de cardinalidad soportados por la restricción del oráculo 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—la duplicación de subárboles aplicada sobre un nodo de texto equivale a concatenarle a dicho nodo su propio su valor, convirtiéndolo en un nodo de texto espurio que produce la violación de la restricción 1.

En principio, el número de salidas incorrectas empleadas en este caso de estudio no es muy amplio para validar la efectividad del oráculo. No obstante, en [4] se ha hecho una evaluación más exhaustiva con un mayor número de programas, y los resultados han sido análogos.

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 de comportamiento del programa bajo prueba suministrados a mano, 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 empleando 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 es capaz de detectar un número de fallos razonable, aun cuando los requisitos de comportamiento del programa bajo prueba 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. También se considerará la posibilidad de mejorar la especificación del oráculo complementándola con información derivada del criterio de selección que sea empleado durante las pruebas.

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 Test4SOA (TIN2007-67843-C06-01).

Referencias

- [1] Abiteboul, S. "Querying Semi-Structured Data." International Conference on Database Theory. 1997. 1-18.
- [2] Barbosa, D., A. Mendelzon, J. Keenleyside, and K. Lyons. "ToXgene: a template-based data generator for XML." International Conference on Management of Data. 2002. 616-616.
- [3] Bertolino, A. "TAXI—A tool for XML-Based Testing." International Conference on Software Engineering, 2007. 53-54.
- [4] Dae S. Kim-Park, Claudio de la Riva, Javier Tuya. "An Automated Test Oracle for XML Processing Programs." International Workshop on Software Test Output Validation, 2010.
- [5] de la Riva, C., J. García-Fanjul, and J. Tuya. "A Partition-Based Approach for XPath Testing." International Conference on Software Engineering Advances. 2006. 17--22.
- [6] Weyuker, E. J. "On Testing Non-testable Programs." The Computer Journal, 25 (4), 1982: 465-470.
- [7] 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).
- [8] World Wide Web Consortium. XML Query Use Cases. 23 de Marzo de 2007. <http://www.w3.org/TR/xquery-use-cases/> (último acceso: 2010).
- [9] World Wide Web Consortium. XQuery 1.0 and XPath 2.0 Data Model (XDM). Enero 23, 2007. <http://www.w3.org/TR/xpath-datamodel/> (último acceso: 2010).
- [10] 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).
- [11] Xu, W., J. Offut, and J. Luo. "Testing Web Services by XML Perturbation." International Symposium on Software Reliability Engineering. 2005. 257-266.