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

Editorial

REICIS

Más allá de las crisis que vivimos, la calidad sigue siendo un elemento fundamental para la competitividad y el desarrollo de la Sociedad. Entendida como la satisfacción de las necesidades expresadas o implícitas de los clientes y usuarios dentro de unas limitaciones o condiciones de tiempo y dinero, la calidad no es negociable. Simplemente no podemos permitirnos no hacer bien las cosas confiando en bajar precios o plantear otras alternativas: no es posible plantearnos esa alternativa. Por supuesto, la calidad no debe estar reñida con la productividad: es más, deben ser la misma cara de una misma actitud. Tampoco debemos confundir esta cualidad con la excelencia. En el campo de la informática, y más precisamente en el del software, no se trata de pensar sólo en hacer bien las cosas sino, muy especialmente, en saber exactamente qué quiere el clientes y el usuario. Esto, que puede parecer muy sencillo, no lo es tanto ya que las estadísticas confirman que un 35% de los problemas en los proyectos de software proceden de defectos en los requisitos. Como vemos, debemos seguir insistiendo en la concienciación y en la implantación de distintas técnicas y estrategias para lograr el desarrollo y la competitividad necesarios, sin descuidar un elemento fundamental: el factor humano, los profesionales y las organizaciones.

Desde REICIS (www.ati.es/reicis) queremos seguir apostando por concienciar de la importancia de la inversión en calidad de software. Por ello, es esencial ofrecer datos y ejemplos claros de aplicación real de las propuestas que se publican en la revista así como innovaciones reales para superar las dinámicas tóxicas e improductivas en el desarrollo de software. Insistiremos también en desarrollar estudios propios o en colaboración con otras organizaciones para lograr una mejor comprensión de las prácticas reales y de cómo pueden ser mejoradas. Así mismo, en colaboración con grupo de calidad del software (www.ati.es/gtcalidadsoft) de ATI, esperamos contribuir a la actualización, la formación y la innovación ofreciendo lo mejor (jornadas, formación, etc.) a todos y al mínimo coste posible, incluso gratuitamente, como esta revista, la documentación en web, nuestra lista de distribución, etc. Esperamos seguir inspirando las mejoras en calidad e ingeniería de software que, también, redunden en mejoras de productividad y en prácticas más razonables y satisfactorias para todos los profesionales del software. Debemos conseguir que los resultados finales de nuestro software sean, también, plenamente satisfactorios y sostenibles.

Luis Fernández Sanz
Director

Presentación

REICIS

Este número de REICIS publica, tras el proceso de revisión de nuestro comité editorial, dos contribuciones extendidas y revisadas, seleccionadas de entre las remitidas al taller PRIS'10 celebrado en Valencia el día 7 de septiembre de 2010.

En este caso, el primero de los trabajos publicados corresponde al titulado "Aplicación de un oráculo de prueba automatizado a la evaluación de salidas de programas basados en XML" procedente de autores de la Universidad de Oviedo. En este artículo, Dae S. Kim-Park, Claudio de la Riva y Javier Tuya presentan una propuesta centrada en las pruebas de aplicaciones que procesan datos XML. Su aportación se centra en plantear un oráculo de prueba automatizado que permita determinar la salida esperada en los casos de prueba para este tipo de software.

El segundo trabajo se titula "Equivalencias entre los operadores de mutación definidos para WS-BPEL 2.0 y los definidos para otros lenguajes" y ha sido elaborado por Juan Boubeta-Puig, Inmaculada Medina-Bulo y Antonio García-Domínguez de la Universidad de Cádiz. Centrado en la técnica de pruebas de nutación, el trabajo presenta una evaluación cualitativa de los operadores de mutación definidos para el lenguaje de procesos de negocio WS-BPEL 2.0 y la comparativa con los definidos para otros lenguajes.

Para completar las contribuciones del taller PRIS 2010, su coordinador Claudio de la Riva de la Universidad de Oviedo nos ofrece una reseña que resume las principales aportaciones del mismo.

Finalmente, en la columna de Actualidad Invitada, Ramón Puigjaner, que es el actual vicepresidente de IFIP además de una figura destacada en la historia de ATI y de la informática en España, nos permite conocer en la columna Actualidad Invitada de este número, gracias a su privilegiada posición en la organización, cuáles son los planes de las principales actividades de la International Federation for Information Processing.

Luis Fernández Sanz

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.

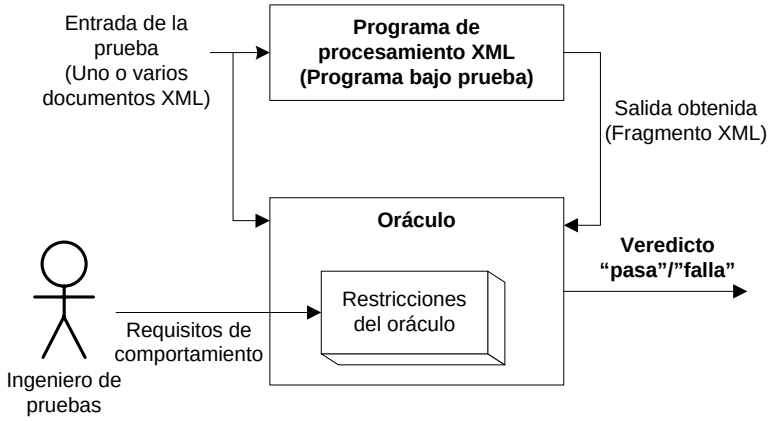

Figura 1. Escenario de prueba para programas de procesamiento XML

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.

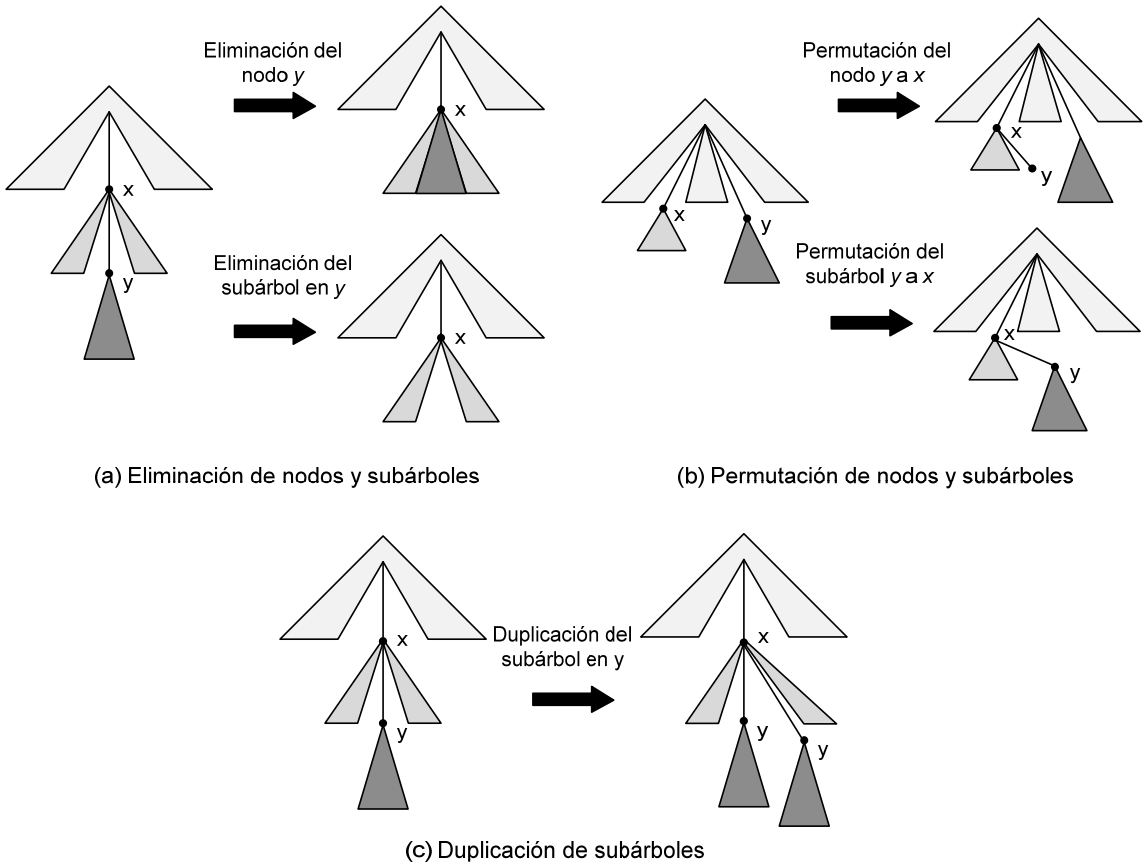

Figura 5. Efecto de las operaciones de perturbación sobre estructuras en forma de árbol (XML).

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).

Equivalencias entre los operadores de mutación definidos para WS-BPEL 2.0 y los definidos para otros lenguajes

Juan Boubeta-Puig, Inmaculada Medina-Bulo y Antonio García-Domínguez
Departamento de Lenguajes y Sistemas Informáticos
Universidad de Cádiz
{juan.boubeta, inmaculada.medina, antonio.garciadominguez}@uca.es

Resumen

Para aplicar prueba de mutaciones a un programa escrito en un lenguaje concreto se requiere un conjunto de operadores de mutación definido para el mismo. Además, es necesario que dichos operadores cubran adecuadamente todas las características del lenguaje para que sean efectivos. En este artículo, se evalúan cualitativamente los operadores de mutación definidos para el lenguaje de ejecución de procesos de negocio WS-BPEL 2.0 y se estudian las analogías y diferencias entre éste y otros lenguajes. Se revisan los operadores existentes para varios lenguajes de propósito general así como para otros específicos del dominio. Los resultados confirman que WS-BPEL es muy diferente a otros lenguajes, ya que aproximadamente sólo la mitad de sus operadores es equivalente a los de otros. Por tanto, es posible mejorar su conjunto de operadores de mutación.

Palabras clave: prueba de software, análisis de mutaciones, servicios web, WS-BPEL.

Equivalences between mutation operators defined for WS-BPEL 2.0 and other languages

Abstract

Applying mutation testing to a program written in a certain language requires that a set of mutation operators is defined for that language. The mutation operators need to adequately cover the features of that language in order to be effective. In this work, we evaluate qualitatively the operators defined for the Web Services Business Process Execution Language 2.0 (WS-BPEL) and study the differences and similarities between WS-BPEL and other languages. We review the existing operators for several structured and object-oriented general-purpose programming languages, and for several domain-specific languages. Results confirm that WS-BPEL is very different from other languages, as near half of the mutation operators for this language are equivalent to those of other languages. Our study concludes that the set of WS-BPEL mutation operators can be improved.

Key words: software testing, mutation analysis, web services, WS-BPEL.

Boubeta-Puig, J., Medina-Bulo, I. y García-Domínguez, A., "Equivalencias entre los operadores de mutación definidos para WS-BPEL 2.0 y los definidos para otros lenguajes", REICIS, vol. 7, no.1, 2011, pp.23-46. Recibido: 15-6-2010; revisado: 12-7-2010; aceptado:30-3-2011

1. Introducción y motivación

La prueba de mutaciones es una técnica de prueba de software basada en fallos, que genera un conjunto de mutantes al aplicar uno o más operadores de mutación. Cada operador realiza un cambio sintáctico en el programa que se desea probar. Los mutantes de primer orden o *First Order Mutants* (FOMs) aplican un único operador por mutante, mientras que los mutantes de orden superior o *Higher Order Mutants* (HOMs) aplican más de uno por mutante [1]. Algunos operadores modelan los errores típicos que suelen cometer los programadores mientras que otros miden ciertos criterios de cobertura.

El lenguaje de ejecución de procesos de negocio o *Web Services Business Process Execution Language* (WS-BPEL) [2] permite a los programadores definir nuevos servicios web o *Web Services* (WS) a partir de otros más simples, esto es, una composición de servicios. Este lenguaje soporta la invocación de WS externos tanto de forma síncrona como asíncrona así como la paralelización de bucles, entre otros. Por ello, es un lenguaje muy potente para ciertas aplicaciones, como la implementación de procesos de negocio basados en flujos de trabajo sobre WS existentes; aunque también presenta nuevos retos para las pruebas.

Estero et al. han definido un conjunto de operadores de mutación para WS-BPEL en [3] y los han evaluado cuantitativamente en [4]. Nuestro trabajo se diferencia de este en que se realiza una comparativa cualitativa entre los operadores definidos para WS-BPEL y los definidos para otros lenguajes. Los objetivos son comprobar si falta por definir alguno para WS-BPEL y establecer las diferencias y analogías entre WS-BPEL y otros lenguajes, ya que en la actualidad existen pocos trabajos sobre estos operadores.

El resto del artículo se estructura de la siguiente forma. En la sección 2 se especifican las características principales de WS-BPEL. En la sección 3 se presentan algunos trabajos que definen operadores de mutación para C, Fortran, Ada, C++, C#, ASP.NET, Java, SQL y XSLT. En la sección 4 se definen los operadores de mutación para WS-BPEL. En las secciones 5, 6, 7 y 8 se describe, respectivamente, qué operadores de mutación para otros lenguajes son comparables a los de WS-BPEL, los operadores de WS-BPEL que no son aplicables a otros lenguajes, los operadores de mutación para otros lenguajes que no son aplicables a WS-BPEL y los que podrían aplicarse a programas WS-BPEL y que, sin embargo, no están definidos para este lenguaje. En la sección 9 se presentan los resultados

obtenidos. Finalmente, en la sección 10 se presentan algunas conclusiones y las líneas de trabajo futuras.

2. El lenguaje WS-BPEL

WS-BPEL es un lenguaje basado en XML que permite implementar procesos de negocio utilizando WS externos. El proceso de negocio resultante es un WS más complejo, conocido como una composición de servicios. Cada proceso WS-BPEL se divide en cuatro secciones:

1. Definición de relaciones con los socios externos, que son el cliente que utiliza el proceso de negocio y los WS a los que llama el proceso.
2. Definición de las variables que emplea el proceso, basada en *Web Services Description Language* (WSDL) y *XML Schema*.
3. Definición de los distintos tipos de manejadores que puede utilizar el proceso: manejadores de fallos (indican las acciones a realizar en caso de producirse un fallo interno o en un WS al que se llama), manejadores de eventos (especifican las acciones a realizar en caso de que el proceso reciba una petición durante su flujo normal de ejecución), manejadores de terminación (gestionan la terminación del proceso) y manejadores de compensación (permiten deshacer una invocación previa de WS).
4. Descripción del comportamiento del proceso de negocio; esto se logra a través de las actividades que proporciona el lenguaje.

Todos los elementos comentados anteriormente son globales por defecto. Sin embargo, también existe la posibilidad de declararlos de forma local mediante el elemento XML *scope*, que permite dividir el proceso de negocio en diferentes ámbitos.

Los principales elementos constructivos de un proceso WS-BPEL son las actividades, que pueden ser de dos tipos: básicas y estructuradas. Las actividades básicas son las que realizan una determinada labor (recepción de un mensaje de un socio externo, envío de un mensaje a un socio externo, etc.). Las actividades estructuradas pueden contener otras actividades y definen la lógica de negocio.

Cada actividad se representa mediante un elemento XML. A las actividades pueden asociarse un conjunto de atributos y un conjunto de contenedores. Estos últimos pueden incluir diferentes elementos, que a su vez pueden tener atributos asociados.

WS-BPEL permite realizar acciones en paralelo y de forma sincronizada. Por ejemplo, la actividad *flow* permite ejecutar un conjunto de actividades concurrentemente especificando las condiciones de sincronización entre ellas. En la figura 1 se muestra un ejemplo en el que *flow* invoca a tres WS en paralelo, comprobarVuelo, comprobarHotel y comprobarAlquilerCoche. Además, existe otro WS, reservarVuelo, que solo se invocará si se completa comprobarVuelo. Esta sincronización entre actividades se consigue estableciendo un enlace o *link*; por lo que la actividad objetivo del enlace se ejecutará solo si se ha completado la actividad fuente de ese enlace.

```
<flow> ← Actividad estructurada
  <links> ← Contenedor
    <link name="comprobarVuelo-reservarVuelo" ← Atributo /> ← Elemento
  </links>
  <invoke name="comprobarVuelo" ... > ← Actividad básica
    <sources> ← Contenedor
      <source linkName="comprobarVuelo-reservarVuelo" />
    </sources>
  </invoke>
  <invoke name="comprobarHotel" ... />
  <invoke name="comprobarAlquilerCoche" ... />
  <invoke name="reservarVuelo" ... >
    <targets> ← Contenedor
      <target linkName="comprobarVuelo-reservarVuelo" /> ← Elemento
    </targets>
  </invoke>
</flow>
```

Figura 1. Ejemplo de una actividad *flow* para WS-BPEL 2.0

En WS-BPEL pueden utilizarse distintos lenguajes de expresiones. Todos los motores WS-BPEL estándar soportan *XML Path Language* 1.1 (XPath). XPath es un lenguaje declarativo que permite realizar consultas sobre documentos XML y que dispone de los operadores aritméticos, relacionales y lógicos, con una sintaxis similar a la de los lenguajes tradicionales.

3. Trabajos relacionados

Se han publicado muchos trabajos sobre la aplicación de la prueba de mutaciones [5]. En este artículo se seleccionan los trabajos que definen los operadores de mutación para los lenguajes de propósito general más conocidos de programación estructurada y orientada a objetos, y para algunos de los más populares específicos del dominio. Todos los operadores comentados en este trabajo se recogen en la tabla 1.

En cuanto a los operadores de mutación definidos para los lenguajes de programación estructurada, Agrawal et al. definen un conjunto de 77 operadores para C [6], que son implementados más tarde en la herramienta Proteum [7], y los comparan con los operadores de mutación definidos para Fortran.

En un trabajo posterior, Barbosa et al. [8] proponen un procedimiento que permite seleccionar sistemáticamente un conjunto de operadores suficientes para C. La técnica de mutación selectiva trata de encontrar un conjunto de operadores de mutación que genere un subconjunto de todos los posibles mutantes sin pérdida significativa de la efectividad de los casos de prueba [5]. Algunos autores han propuesto los operadores de mutación suficientes para un lenguaje específico. Sin embargo, la mayoría de los estudios se han realizado sobre los operadores de mutación tradicionales.

Los 10 operadores de mutación propuestos como suficientes a partir de los 77 operadores para C son: SWDD (sustituye *while* por *do-while*), SMTC (introduce una guarda antes del cuerpo de un bucle), SSDL (elimina una sentencia), OLBN (sustituye un operador lógico por un operador de bit), OASN (sustituye un operador aritmético por uno de desplazamiento), ORRN (sustituye un operador relacional por otro), VTWD (incrementa o decrementa en 1 el valor de una variable escalar), VDTR (proporciona cobertura de dominio para variables escalares), Cccr (sustituye una constante por otra) y Ccsr (sustituye una constante por un escalar).

King y Offutt definen un conjunto de 22 operadores de mutación para Fortran y los implementan en la herramienta Mothra [9]. Más tarde, Offutt et al. [10] enumeran los 5 operadores que son suficientes: ABS (antes de una expresión o subexpresión, inserta el operador unario de valor absoluto, valor absoluto negativo o el que comprueba si la expresión o subexpresión es cero), UOI (inserta un operador unario), LCR (sustituye un

operador lógico por otro), AOR (sustituye un operador aritmético por otro) y ROR (sustituye un operador relacional por otro).

Nombre	*Lenguaje(s) y referencia(s)*	*Sección*	*Nombre*	*Lenguaje(s) y referencia(s)*	*Sección*
ABS	Fortran [9]; Java [24]; SQL [25]	3	OLBN	C [6]	3
AOR	Fortran [9]; C++ [12]; Java [19], [24]; SQL [25]; XSLT [27]	3	OLLN	C [6]	5.2
CBD	Java [23]	5.4	OIPM	C [6]	7
Cccr	C [6]	3	ORO	ASP.NET [16]	5.2
Ccsr	C [6]	3	ORRN	C [6]	3
CMC	Ada [11]	3	OVV	Ada [11]; C++ [12]	5.1
CRCR	C [6]	5.2	PPD	Java [18]	3
CRP	Fortran [9]	5.2	ROR	Fortran [9]; Java [19]; SQL [25]; XSLT [27]	3
DNR	XSLT [27]	8	SAN	Fortran [9]	5.3
DSI	XSLT [27]	8	SDL	Fortran [9]	5.3
EAI	Ada [11]	3	SDWD	C [6]	5.3
EDT	Ada [11]	5.2	SEE	Ada [11]	5.3
EEO	Ada [11]	3	SER	Ada [11]	5.4
EEU	Ada [11]	3	SMO	ASP.NET [16]	5.3
EEZ	Ada [11]	3	SMTC	C [6]	3
EHR	C# [14]; Java [28]	5.4	SMVB	C [6]	7
ELR	Ada [11]	3	SRN	Ada [11]	5.3
EMO	ASP.NET [16]	5.2	SRSR	C [6]	7
EMR	Ada [11]	3	SRW	Ada [11]	5.3
ENI	Ada [11]	3	SSDL	C [6]	3
EOA	Java [18]	3	SSWM	C [6]	7
EOR	Ada [11]	3	STRP	C [6]	5.3
ERR	Ada [11]	3	SVR	Fortran [9]	5.1
ESR	Ada [11]	3	SWDD	C [6]	3
EUI	Ada [11]	3	SWR	Ada [11]	5.3
EUR	Ada [11]	3	UOD	Java [24]	5.2
IBO1	C++ [12]	5.2	UOI	Fortran [9]; C++ [12]; Java [24]; SQL [25]	3
IBO2	C++ [12]	5.2	Uuor	C [6]	8
IBO3	C++ [12]	5.2	VDTR	C [6]	3
IHD	Java [18]	3	VGAR	C [6]	7
IRP	SQL [25]	5.1	VGSR	C [6]	5.1
ISD	Java [18]	3	VGTR	C [6]	7
LCR	Fortran [9]; SQL [25]; Java [20] (como COR), [24]; XSLT [27]	3	VLSR	C [6]	5.1
MXT	Java [22]	5.2	VR	XSLT [27]	5.1
OAAN	C [6]	5.2	VTWD	C [6]	3
OASN	C [6]	3	XER	XSLT [27]	5.2

Tabla 1. Índice de los operadores de mutación para otros lenguajes comentados en este estudio.

Por otro lado, Offutt et al. definen 65 operadores de mutación para Ada [11] y realizan una comparativa entre los operadores definidos para Ada, C y Fortran. Además,

proponen CMC (cobertura de condición múltiple) como el único operador suficiente de mutación de cobertura y 12 operadores suficientes de expresiones: EAI (inserta el valor absoluto), ENI (inserta el valor absoluto negativo), EEZ (inserta el subprograma *Except_on_Zero* antes de cada expresión y subexpresión aritmética), EOR (sustituye un operador aritmético por otro), ERR (sustituye un operador relacional por otro), EMR (sustituye cada IN por NOT IN y viceversa), ELR (sustituye un operador lógico por otro), EUI (inserta el operador menos unario), EUR (sustituye el operador unario por otro), ESR (sustituye cada nombre de función y subrutina por otro que tenga la misma signatura y pertenezca al mismo paquete), EEO (inserta el subprograma *Except_on_OverFlow* antes de cada expresión aritmética) y EEU (inserta el subprograma *Except_on_UnderFlow* antes de cada expresión aritmética).

Otros autores definen operadores de mutación para lenguajes de programación orientados a objetos como C++, C#, ASP.NET o Java.

Zhang [12] define 24 operadores para C++. Dereziňska define 40 operadores de mutación para C# en [13] y [14], que son implementados en la herramienta CREAM [15]. Mansour y Houri [16] han propuesto un conjunto de 15 operadores de mutación para evaluar la calidad de las pruebas realizadas a aplicaciones web escritas en ASP.NET.

Existen muchos trabajos sobre operadores de mutación para Java. Ma et al. definen 26 operadores de mutación de clases [17] (redefinidos en trabajos posteriores como en [18]) y de métodos [19]; estos operadores han sido integrados en la herramienta MuJava [20]. Más tarde, estudian la efectividad de los operadores de mutación de clases que producen pocos mutantes e identifican los siguientes [21]: IHD (elimina una variable oculta), ISD (elimina la palabra reservada *super*), PPD (declara un parámetro variable con un tipo de clase hija) y EOA (sustituye una asignación de referencia y una asignación de contenido). Estos autores planean ampliar su estudio para determinar un conjunto de operadores de mutación suficientes.

Bradbury et al. [22] definen 24 operadores de mutación específicos para el comportamiento concurrente de Java. Ji et al. [23] han definido 5 operadores para la captura de excepciones. Madeyski y Radyk [24] han seleccionado 16 operadores de mutación de entre todos los definidos por Ma et al. [17] y Offutt et al. [18], y los han integrado en la herramienta Judy [24]. Otros trabajos definen operadores de mutación para SQL, un

lenguaje específico de dominio para operar sobre bases de datos relacionales. Tuya et al. [25] definen 22 operadores suficientes para consultas SELECT, integrados en la herramienta SQLMutation [26], que son evaluados posteriormente por Dereziňska [1].

Lonetti y Marchetti [27] proponen 21 operadores de mutación para *Extensible Stylesheet Language Transformations* (XSLT), un lenguaje que permite transformar un documento XML en otros formatos. Los autores han adaptado al lenguaje XSLT algunos de los operadores de variables y expresiones implementados en las herramientas Mothra [9] y MuJava [20]. Además, definen operadores específicos para la manipulación de elementos XSLT, como expresiones XPath y reglas de plantillas. Una colección de reglas es un programa XSLT básico.

4. Operadores de mutación para WS-BPEL

Estero et al. [3] han definido 26 operadores de mutación para WS-BPEL (ver tabla 2). Los nombres de estos operadores están compuestos por tres letras en mayúsculas. La primera letra identifica su categoría. Existen cuatro categorías, de acuerdo con el tipo de elemento sintáctico de WS-BPEL afectado por el operador: mutación de Identificadores (I), de Expresiones (E), de Actividades (A), y de condiciones eXcepcionales y eventos (X). Las últimas dos letras representan la acción realizada sobre el elemento sintáctico.

Es importante tener en cuenta que estos operadores solo reemplazan o eliminan código: ninguno añade código. Los autores consideran que es complicado añadir código por error, puesto que los procesos WS-BPEL son codificados normalmente mediante herramientas gráficas (excepto las expresiones XPath). Cabe destacar que, aunque se usen herramientas gráficas, los programadores podrían escoger el elemento equivocado seleccionando, por ejemplo, una actividad *while* en lugar de una actividad *repeatUntil*, o estableciendo el valor de un atributo a *yes* en vez de *no*.

5. Operadores equivalentes a los de WS-BPEL

En esta sección se estudian las similitudes existentes entre los operadores definidos para WS-BPEL (ver sección 4) y los definidos para otros lenguajes (ver sección 3).

La tabla 3 resume los resultados presentados en esta sección. Los operadores de mutación definidos para otros lenguajes que son equivalentes a los de WS-BPEL se han

clasificado en las cuatro categorías mencionadas en la sección 4 y se han distinguido dos tipos de operadores: similares y completamente equivalentes.

Operador	*Descripción*
Mutación de Identificadores	
ISV	Sustituye el identificador de una variable por el de otra del mismo tipo
Mutación de Expresiones	
EAA	Sustituye un operador aritmético (+, -, *, *div*, *mod*) por otro del mismo tipo
EEU	Elimina el operador menos unario de cualquier expresión
ERR	Sustituye un operador relacional (<, >, >=, <=, =, !=) por otro del mismo tipo
ELL	Sustituye un operador lógico (*and*, *or*) por otro del mismo tipo
ECC	Sustituye un operador de camino (/, //) por otro del mismo tipo
ECN	Modifica una constante numérica incrementando o decrementando su valor en una unidad, añadiendo o eliminando un dígito
EMD	Modifica una expresión de duración cambiando por 0 o por la mitad el valor inicial
EMF	Modifica una expresión de fecha límite cambiando por 0 o por la mitad el valor inicial
Mutación de Actividades	
Relacionados con la concurrencia	
ACI	Cambia el atributo *createInstance* de las actividades de recepción de mensajes a no
AFP	Cambia una actividad *forEach* secuencial a paralela
ASF	Cambia una actividad *sequence* por una actividad *flow*
AIS	Cambia el atributo *isolated* de un *scope* a no
No concurrentes	
AEL	Elimina una actividad
AIE	Elimina un elemento *elseif* o el elemento *else* de una actividad *if*
AWR	Cambia una actividad *while* por una actividad *repeatUntil* y viceversa
AJC	Elimina el atributo *joinCondition* de cualquier actividad en la que aparezca
ASI	Intercambia el orden de dos actividades hijas de una actividad *sequence*
APM	Elimina un elemento *onMessage* de una actividad *pick*
APA	Elimina el elemento *onAlarm* de una actividad *pick* o de un manejador de eventos
Mutación de Condiciones Excepcionales y Eventos	
XMF	Elimina un elemento *catch* o el elemento *catchAll* de un manejador de fallos
XMC	Elimina la definición de un manejador de compensación
XMT	Elimina la definición de un manejador de terminación
XTF	Cambia el fallo lanzado por una actividad *throw*
XER	Elimina una actividad *rethrow*
XEE	Elimina un elemento *onEvent* de un manejador de eventos

Tabla 2. Operadores de mutación para WS-BPEL 2.0

Los operadores similares son aquellos operadores definidos para otros lenguajes que deben redefinirse con pequeños cambios para que sean equivalentes a otro operador de mutación para WS-BPEL (aparecen marcados con ⋆ en la tabla 3), mientras que los operadores completamente equivalentes, como su nombre indica, son aquellos que no necesitan ser redefinidos para ser comparables con los de WS-BPEL.

		Lenguajes de programación estructurada			*Lenguajes de programación orientada a objetos*				*Lenguajes específicos del dominio*	
Categoría	*WS-BPEL*	*C*	*Fortran*	*Ada*	*C++*	*C#*	*ASP .NET*	*Java*	*SQL*	*XSLT*
Mutación de identificador	ISV	⋆VGSR, ⋆VLSR	SVR	OVV	OVV				⋆IRP	⋆VR
Mutación de expresión	EAA	OAAN	AOR	EOR	AOR, IBO1		⋆ORO	AOR	AOR	AOR
	EEU							UOD		
	ERR	ORRN	ROR	ERR	IBO2		⋆ORO	ROR	ROR	ROR
	ELL	OLLN	LCR	ELR	IBO3		⋆ORO	LCR	LCR	LCR
	ECC									⋆XER
	ECN	⋆CRCR	⋆CRP	⋆EDT			⋆EMO			
	EMD							⋆MXT		
Mutación de actividad	AEL	⋆SSDL	⋆SDL	⋆SRN			⋆SMO			
	AIE	⋆STRP	⋆SAN	⋆SEE			⋆SMO			
	AWR	⋆SWDD, ⋆SDWD		⋆SWR, ⋆SRW						
Mutación de condición excepcional y evento	XMF					⋆EHR		⋆EHR, ⋆CBD		
	XTF			⋆SER						

Tabla 3. Equivalencias entre los operadores de mutación para WS-BPEL y otros lenguajes.

5.1. Mutación de identificadores

Existen operadores de sustitución de identificadores para la mayoría de los lenguajes de programación estudiados, excepto para C#, ASP.NET y Java.

Para el lenguaje C existen dos operadores similares a ISV: VGSR y VLSR. VGSR cambia los identificadores de las variables escalares que son globales a una función, mientras que VLSR cambia variables escalares que son locales a una función. Las funciones no existen en WS-BPEL, pero el ámbito de una declaración de variable puede limitarse introduciendo su declaración dentro de un elemento *scope*. VGSR podría ser equivalente si se limita ISV a que sustituya un identificador por otro que se encuentre en los ámbitos padres, excepto en el más próximo. VLSR podría ser equivalente si ISV únicamente sustituye un identificador por otro del ámbito padre más próximo.

IRP, para consultas SELECT de SQL, tiene cierto parecido con ISV. Este operador sustituye cada parámetro de consulta por otro parámetro, columna o constante de tipo compatible. Es más general que ISV, puesto que sustituye identificadores por constantes y

no solo por otros identificadores. Al igual que hace ISV, comprueba el tipo de la sustitución para evitar la generación de mutantes inválidos.

El operador VR para XSLT es similar a ISV. VR sustituye el valor del atributo *select* del elemento *variable* de XSLT por otro valor de los atributos *select* contenidos en otros elementos *variable*, mientras que ISV sustituye el valor del atributo *variable* de las actividades WS-BPEL.

Los operadores SVR para Fortran, OVV para Ada y OVV para C++ son equivalentes a ISV sin realizar ninguna modificación sobre sus definiciones.

5.2. Mutación de expresiones

A continuación, se tratarán las mutaciones de operadores aritméticos, relacionales y lógicos, así como las mutaciones de constantes y de duración.

OAAN para C es equivalente a EAA. Para Ada, EOR muta, además de los operadores aritméticos básicos, el operador de exponenciación ** y el de resto *rem*.

El operador AOR, originalmente para Fortran y más tarde adaptado para SQL, Java, C++ y XSLT, sustituye cada uno de los operadores (+, –, *, /) por otro operador del mismo conjunto. Este operador es equivalente a EAA. Las versiones para SQL y Fortran también pueden sustituir la expresión por *leftop* (devuelve el operando izquierdo, ignorando el derecho) y *rightop* (devuelve el operando derecho, ignorando el izquierdo). Además, la versión para Fortran puede reemplazar el operador por la función módulo MOD y el operador de exponenciación **.

Hay otro operador para C++ que es equivalente a EAA: IBO1. Este sustituye cada operador multiplicativo (*, /, %) o aditivo (+, –) por otro de los operadores multiplicativos o aditivos, respectivamente.

ORO es el operador de mutación de expresiones para ASP.NET. Este sustituye un operador por otro; puesto que no define cuál debe ser el tipo de operador, se considera similar a EAA, ERR y ELL.

Tan solo se ha encontrado un operador equivalente a EEU: UOD para Java. La mayoría de los autores prefieren insertar el operador de negación unario o reemplazarlo.

Los operadores equivalentes a ERR son: ORRN para C, ROR para Fortran, SQL, Java y XSLT, ERR para Ada y IBO2 para C++. En el caso de SQL y Fortran, la expresión

relacional también se sustituye por *trueop* (siempre devuelve un valor booleano verdadero) y *falseop* (siempre devuelve un valor booleano falso).

ELL tiene cuatro operadores de mutación equivalentes: OLLN para C, LCR para Fortran, SQL, Java y XSLT, ELR para Ada y IBO3 para C++. En el caso de SQL y Fortran, los operadores lógicos también pueden ser sustituidos por *falseop*, *trueop*, *leftop* y *rightop*. Además de los operadores *and* y *or*, ELR también cambia el operador lógico por *xor* y las formas en corto circuito *or else* and *and then*.

ECC sustituye un operador de camino (/, //) por otro del mismo tipo. De entre los lenguajes estudiados, los operadores de camino que permiten recorrer árboles aparecen únicamente en XPath. XER, para XSLT, sustituye una expresión XPath del atributo *select* del elemento *value-of* por una expresión XPath contenida en otro atributo *select* del elemento *value-of*. Este es similar a ECC si solo reemplaza los de camino.

El operador de mutación para C equivalente a ECN es CRCR. Este asegura que se manejan apropiadamente las referencias escalares de enteros y flotantes, como no hace ECN, que considera todos los números iguales, conforme al sistema de tipos XPath. CRCR puede incrementar, decrementar y sustituir una referencia de puntero a *null*. Sin embargo, CRCR no añade o elimina dígitos a una constante como sí hace ECN.

Otros operadores comparables con ECN son: EDT para Ada (si se considera únicamente mutaciones de expresiones constantes), EMO para ASP.NET y CRP para Fortran. CRP también permite incrementar o decrementar constantes de precisión doble en un 10%, y sustituir 0 por 0,01 y -0,01.

MXT para Java es el único operador que se ha encontrado para modificar expresiones de duración, al igual que hace EMD. Este operador para Java modifica el argumento opcional de las llamadas a los métodos estándar de Java *wait()*, *sleep()*, *join()* y *await()*, que representa el intervalo en milisegundos que el hilo actual debería esperar. Mientras que MXT cambia el tiempo especificado por el doble o la mitad, EMD lo cambia por 0 o la mitad. La duración es un valor entero integrado en Java, mientras que en WS-BPEL se utiliza una constante de duración de XML Schema.

5.3. Mutación de actividades

Algunos de los operadores de mutación de actividades para WS-BPEL modelan el fallo que puede cometerse al elegir una actividad que no es la más adecuada para las acciones que se

deben realizar, sustituyendo una actividad por otra. Los demás modelan la elección de un valor incorrecto para los atributos de las actividades, sustituyendo para ello su valor actual por otro valor válido. Estos operadores se clasifican en dos tipos, los relacionados con la concurrencia y los no concurrentes.

SSDL para C, SDL para Fortran, SRN para Ada y SMO para ASP.NET son similares a AEL. Mientras AEL elimina una actividad, los otros operadores eliminan una sentencia. No obstante, aunque SRN sustituye una sentencia por NULL, puede considerarse similar a AEL. STRP para C, SAN para Fortran, SEE para Ada y SMO para ASP.NET son similares a AIE, pero SMO difiere en que solo elimina una única sentencia de la rama *else*, en lugar de la rama entera.

Existen dos operadores para C que son similares a AWR: SWDD y SDWD. Estos operadores cambian un bucle *while* por un bucle *do-while* y viceversa, respectivamente. Asimismo, AWR intercambia las actividades *while* y *repeatUntil*, que son semánticamente equivalentes a las sentencias anteriores.

SWR y AWR, para Ada, también intercambian dos sentencias pero, en este caso, cambia el bucle *while* condicional por el bucle *loop* incondicional y viceversa.

5.4. Mutación de excepciones y eventos

Los operadores de mutación relacionados con las condiciones excepcionales y eventos para WS-BPEL están relacionados con los distintos tipos de manejadores que proporciona WS-BPEL: manejadores de fallos, eventos, compensación y terminación.

Los manejadores de fallos de WS-BPEL utilizan elementos *catch* para manejar fallos específicos y el elemento *catchAll* para manejar el resto. Estos son bastante similares a las excepciones, que son frecuentemente utilizadas para manejar los errores en los lenguajes modernos de programación orientada a objetos. Solo se han encontrado dos operadores similares a XMF: EHR para Java y C#, y CBD para Java. EHR elimina un único bloque *catch* junto con el bloque *finally* si no queda ningún bloque *catch*. CBD simplemente elimina un bloque *catch*.

XTF cambia el nombre del fallo lanzado por una actividad *throw* por otro del mismo ámbito. Esta actividad permite lanzar un fallo determinado, cuyo nombre se especifica mediante el atributo *faultName*. En el caso de los lenguajes tradicionales, la sentencia *throw*

tiene un comportamiento similar. SER, para Ada, cambia el nombre de la excepción utilizada en una sentencia *raise*.

6. Operadores no aplicables a WS-BPEL

En esta sección, se enumeran los operadores definidos para WS-BPEL que no son aplicables a ninguno de los descritos en la sección 3 y se clasifican en dos grupos, atendiendo a las características que mutan: características específicas del lenguaje WS-BPEL y características que comparten WS-BPEL y otros lenguajes.

6.1. Características específicas del lenguaje WS-BPEL

WS-BPEL tiene algunas características específicas que no presentan los otros lenguajes. Por tanto, se necesitan operadores específicos para WS-BPEL que muten dichas características.

ACI cambia el atributo *createInstance* de una actividad *receive* o *pick* de *yes* a *no* solo si el proceso tiene más de una actividad con el atributo a *yes*. Este operador no puede ser aplicado a otros lenguajes porque ninguno de los lenguajes estudiados tiene una sentencia especializada que reciba mensajes provenientes de una red.

AFP sustituye una actividad secuencial *forEach* por una paralela, cambiando el atributo *parallel* de *no* a *yes*. No hay ningún operador para otros lenguajes equivalente a AFP, debido a que no incluyen soporte para ejecutar bucles en paralelo. Para implementar los bucles en paralelo es necesario utilizar bibliotecas de terceras partes o extensiones de lenguajes específicos.

AIS cambia el atributo *isolated* de un elemento *scope* de *yes* a *no*. Si *isolated* tiene el valor *yes*, las lecturas y escrituras de variables concurrentes dentro del *scope* son serializadas para preservar la consistencia de los datos. Si se establece el valor *no* a *isolated* podría obtenerse un comportamiento inesperado que debería ser detectado con los casos de prueba existentes. La mayoría de los lenguajes estudiados permiten limitar el ámbito de una declaración introduciendo un constructor específico (como un *block* en C y sus descendientes). Sin embargo, no disponen de constructores específicos para serializar el acceso concurrente de un conjunto de variables.

AJC elimina el atributo *joinCondition* de una actividad dentro de un *flow*. Si no se especifica lo contrario, todas las actividades en un *flow* se ejecutan concurrentemente y

comienzan a la vez. Si una actividad *A* necesita comenzar antes que otra *B*, el programador debe crear un enlace o *link* de *A* a *B*. Todas las actividades esperan hasta que sus enlaces de entrada sean actualizados con la información del éxito o fracaso de sus actividades fuentes. Una vez que esta información esté disponible, cada actividad evaluará su condición de unión para decidir si ejecutar o lanzar un fallo. Por defecto, la ejecución continuará si al menos uno de los enlaces de entrada informa de un estado exitoso. El usuario puede personalizar este comportamiento proporcionando la expresión booleana XPath apropiada en el atributo *joinCondition* de una actividad. La sincronización basada en enlaces y el manejo de fallos no están en ninguno de los otros lenguajes estudiados.

APM elimina un elemento *onMessage* de una actividad *pick* si hay más de uno. *pick* se utiliza cuando se espera uno de los mensajes de las actividades previas. En cierta manera, es similar a una sentencia *switch* de C, pero los casos a probar son más complejos. Cada elemento hijo *onMessage* maneja un tipo de mensaje de entrada específico, y cada *onAlarm* gestiona la situación en la que ningún mensaje ha llegado durante el tiempo especificado. Puesto que no existe ninguna estructura de control en los otros lenguajes, se concluye que no hay ningún operador equivalente a APM.

APA elimina el elemento *onAlarm* de una actividad *pick* o de un manejador de eventos. Por la misma razón que se ha comentado anteriormente, tampoco existen operadores equivalentes a APA.

XMC elimina la definición de un manejador de compensación. WS-BPEL lo sustituirá por el manejador de compensación por defecto. Estos son específicos de WS-BPEL: deshacen las invocaciones previas de servicios cuando se lanza un fallo.

XMT elimina la definición de manejador de terminación. WS-BEL lo sustituirá por el manejador de terminación por defecto. Estos manejadores también son específicos de WS-BPEL: contienen las actividades que decidirán si es necesario terminar la ejecución de un *scope*.

XEE elimina un elemento *onEvent* de un manejador de eventos, modelando la situación en la que el programador olvida tratar un evento en particular. Estos manejadores de eventos se implementan manualmente mediante código en los otros lenguajes, ya que no están integrados.

6.2. Características comunes en WS-BPEL y otros lenguajes

A continuación, se enumeran los operadores de mutación definidos para WS-BPEL que mutan características que comparten tanto el lenguaje WS-BPEL como los demás.

EMF modifica una expresión de fecha. Puede aplicarse a la actividad *wait* y al elemento *onAlarm* de la actividad *pick*. Los cambios introducidos son equivalentes a EMD. Sin embargo, no se ha encontrado ningún operador para los otros lenguajes equivalente a EMF.

ASF sustituye una actividad *sequence* por una actividad *flow*. La actividad *sequence* ejecuta secuencialmente las actividades que contiene, mientras la actividad *flow* las ejecuta en paralelo. No existe ningún operador para los otros lenguajes estudiados que cambie la ejecución secuencial de sentencias a paralela.

ASI intercambia el orden de dos actividades hijas de una actividad *sequence*. Tampoco existe ningún operador para los otros lenguajes estudiados que intercambie el orden de dos sentencias que se ejecuten secuencialmente.

XER elimina una actividad *rethrow*. Esta actividad lanza un fallo capturado previamente, por lo que se activarán los manejadores de fallo padres. Ninguno de los operadores para los otros lenguajes elimina el relanzamiento de una excepción.

7. Operadores no aplicables a WS-BPEL

En esta sección se comentan algunos operadores de mutación definidos para los otros lenguajes que no son aplicables a WS-BPEL, con algunos ejemplos.

Algunos operadores para C y los conceptos con los que se relacionan no están disponibles en WS-BPEL 2.0 como, por ejemplo: los operadores de desplazamiento (OASN), los operadores de bits (OLBN), los vectores y registros (VGAR, muta referencias de vectores utilizando vectores globales, y VGTR, muta referencias de estructuras utilizando referencias de estructuras globales), los punteros (OIPM que muta el operador de indirección) y las funciones (SRSR, sustituye una sentencia *return*).

Por otro lado, las sentencias de control de selección múltiple en WS-BPEL 2.0 se encuentran limitadas a la actividad basada en alarmas y mensajes, y a los manejadores de eventos: la construcción *switch* que comprueba valores enteros no está disponible. Por ello, SSWM (muta una sentencia *switch*) tampoco podrá aplicarse a WS-BPEL.

Puesto que los procesos de negocio WS-BPEL se escriben en XML, en lugar de utilizar las llaves ({, }) para estructurar el código se emplean elementos XML. Los operadores de mutación como SMVB para C, que desplaza una llave arriba o abajo por una sentencia, no tiene sentido en WS-BPEL.

Otra diferencia importante es que mientras la mayoría de los operadores mutan sentencias e instrucciones, los de WS-BPEL mutan actividades, elementos y atributos.

Ninguno de los operadores de mutación de clase para Java definidos en [21] puede aplicarse a código WS-BPEL porque mutan características específicas a la orientación a objetos como la herencia y el polimorfismo, así como características específicas de Java no presentes en el lenguaje WS-BPEL.

Tanto WS-BPEL 2.0 como Java permiten manejar la concurrencia, por lo que se podría pensar que los operadores de mutación para la concurrencia de *Java 2 Standard Edition* (J2SE) 5.0 definidos en [22] podrían ser aplicables a WS-BPEL 2.0. Sin embargo, esto no es posible, excepto para el operador XMT, debido a que sus enfoques para soportar concurrencia son bien distintos.

La mayoría de los operadores de mutación definidos para SQL tampoco son aplicables a código WS-BPEL, excepto los de expresiones aritméticas, lógicas, relacionales y unarias. La razón fundamental es que el propósito de estos dos lenguajes es distinto: SQL es un lenguaje de consulta y WS-BPEL es un lenguaje de ejecución de procesos de negocio.

Los operadores definidos para XSLT que mutan plantillas no son equivalentes a ninguno de los operadores para WS-BPEL porque este no dispone de plantillas. Algunos han integrado XSLT como un lenguaje de expresión adicional para WS-BPEL, pero no es una extensión estándar.

8. Operadores para otros lenguajes que podrían aplicarse a WS-BPEL

En esta sección se pretende encontrar otros operadores de mutación definidos para los otros lenguajes estudiados que podrían aplicarse a WS-BPEL y que, sin embargo, Estero et al. no hayan definido. Para comprobar si el conjunto de dichos operadores puede mejorarse, se ha prestado especial atención a los operadores identificados por la técnica de mutación selectiva que modelan los fallos cometidos por los programadores (consúltese dichos operadores en la sección 3).

Los 5 operadores suficientes para Fortran [10] podrían aplicarse a programas WS-BPEL. Sin embargo, no existe ningún operador definido para WS-BPEL que sea equivalente a ABS o UOI. Por tanto, se proponen dos nuevos operadores de mutación de expresiones para WS-BPEL: EAI (inserta el operador de valor absoluto) y EUI (inserta el operador menos unario). Como se muestra en la tabla 4, estos operadores son completamente equivalentes a los de Ada, mientras que los definidos para el resto son similares, ya que realizan otras mutaciones además de las descritas.

Puesto que no se tiene conocimiento de que exista algún trabajo que defina operadores suficientes para XSLT, se han escogido los operadores de mutación para XSLT propuestos en [27]. Como se comentó en la sección 4, los procesos WS-BPEL son normalmente codificados con herramientas gráficas. Sin embargo, las expresiones XPath sí se escriben a mano, por lo que el programador podría incluir código extra por error. Dos de estos operadores que mutan expresiones XPath podrían aplicarse a programas WS-BPEL: DSI, que inserta una doble barra (//) al principio de una expresión XPath, y DNR, que sustituye . (indica el nodo actual) por .. (indica el nodo padre) y viceversa. Por ello, se proponen otros dos operadores para WS-BPEL completamente equivalentes a los dos anteriores: ECI y ENC, respectivamente.

		Lenguajes de programación estructurada			*Lenguajes de programación orientada a objetos*				*Lenguajes específicos del dominio*	
Categoría	*WS-BPEL*	*C*	*Fortran*	*Ada*	*C++*	*C#*	*ASP .NET*	*Java*	*SQL*	*XSLT*
Mutación de expresión	EAI	⋆VDTR	⋆ABS	EAI			⋆EMO	⋆ABS	⋆ABS	
	EUI	⋆Uuor	⋆UOI	EUI	⋆UOI		⋆EMO	⋆UOI	⋆UOI	
	ENC									DNR
	ECI									DSI

Tabla 4. Nuevos operadores propuestos para WS-BPEL junto con sus equivalentes para otros lenguajes.

9. Resultados

En este estudio se ha realizado una comparativa cualitativa entre los operadores de mutación definidos para WS-BPEL y los de otros lenguajes. Mientras que Estero et al. [3] estudiaron las características específicas de este lenguaje para definir los operadores que las mutaran, en este trabajo se ha llevado a cabo el proceso contrario: se han revisado todos los operadores de mutación definidos para los lenguajes de programación estructurada y

orientada a objetos así como específicos del dominio, para comprobar si estos autores han olvidado definir alguno de los operadores para WS-BPEL.

Como puede observarse en la tabla 3, se ha determinado que el operador de mutación XER para XSLT es similar a ECC, el operador XMT para Java es similar a EMD y el operador SER para Ada es similar a XTF; en cambio, Estero et al. los han definido como operadores específicos para WS-BPEL.

En la sección 8 se ha determinado que el conjunto de operadores de mutación para WS-BPEL no es completo y se han añadido 4 operadores de mutación de expresiones: EAI, EUI, ENC y ECI. Por tanto, existe un conjunto de 30 operadores definidos para WS-BPEL. A partir de las tablas 3 y 4 se deduce que 17 (56,7%) de los 30 operadores para WS-BPEL son equivalentes a algún operador definido para los otros lenguajes.

Al clasificar los operadores en las categorías de los operadores WS-BPEL (ver sección 4) se observa que el 91,7% de los operadores de mutación de expresiones para WS-BPEL es equivalente a los de otros lenguajes (ver tabla 5). Este alto porcentaje de similitud puede atribuirse al uso de XPath, que ha heredado algunas características de los lenguajes tradicionales.

Tipo de lenguaje	*Lenguaje*	*Completamente equivalente (%)*	*Similar (%)*	*Total (%)*
Programación estructurada	C	17,6	41,2	58,8
	Fortran	23,5	29,4	52,9
	Ada	35,3	29,4	64,7
Programación orientada a objetos	C++	23,5	5,9	29,4
	C#	0	5,9	5,9
	ASP.NET	0	47	47
	Java	29,4	17,6	47
Específico del dominio	SQL	17,6	17,6	35,2
	XSLT	29,4	11,8	41,2
Categoría				*Total (%)*
Mutación de identificador				100
Mutación de expresión				91,7
Mutación de actividad				27,3
Mutación de condición excepcional y evento				33,3
Todos los lenguajes y categorías				56,7

Tabla 5. Proporciones de los operadores de mutación para WS-BPEL equivalentes a los de otros lenguajes.

El 27,3% de los operadores de mutación de actividades para WS-BPEL es equivalente a los de otros lenguajes. Sin embargo, no se incluye ningún operador

relacionado con la concurrencia (ver tabla 3). Esto se debe a que el enfoque para soportar concurrencia es bastante diferente.

Además, solo el 33,3% de los operadores de mutación de condiciones excepcionales y eventos para WS-BPEL es equivalente a uno de los operadores para otros lenguajes. Esta situación es lógica porque mutan manejadores que son específicos de WS-BPEL, por lo que no están disponibles en otros lenguajes.

Si atendemos a las proporciones de los operadores de mutación para WS-BPEL que son equivalentes a los de otros lenguajes, clasificados según el tipo de lenguaje del que se trata, la tabla 5 muestra que los mayores porcentajes se asocian a los lenguajes de programación estructurada, especialmente a Ada. Por el contrario, el porcentaje de operadores para C# equivalente a los de WS-BPEL es el más bajo. Por consiguiente, se puede concluir que WS-BPEL se asemeja más a los lenguajes de programación estructurada. Además, puede comprobarse que ninguno de los operadores de mutación para ASP.NET y C# es completamente equivalente a un operador para WS-BPEL y deben ser redefinidos con pequeños cambios para que sean equivalentes.

10. Conclusiones y trabajo futuro

En este trabajo se ha evaluado cualitativamente el conjunto de los operadores de mutación definidos por Estero et al. [3] [4] para WS-BPEL 2.0, y se ha comparado con los operadores de mutación definidos para los lenguajes tradicionales más conocidos. En concreto, se han incluido en el estudio los lenguajes de programación estructurada C, Fortran y Ada, los lenguajes de programación orientada a objetos C++, C#, ASP.NET y Java, y los lenguajes específicos del dominio SQL y XSLT.

Se han clasificado los operadores de mutación en cuatro categorías: operadores disponibles tanto para los lenguajes tradicionales como para WS-BPEL, operadores solo aplicables a WS-BPEL, operadores solo aplicables a los lenguajes tradicionales, así como los operadores para otros lenguajes que podrían aplicarse a programas WS-BPEL y que no están ya incluidos en el conjunto de operadores para dicho lenguaje.

A partir de las tablas 3 y 4 se concluye que solo 17 (56,7%) de los 30 operadores para WS-BPEL están disponibles en los lenguajes tradicionales. Esto confirma que WS-BPEL 2.0 tiene algunas características (y tipos de errores al utilizarlas) que son específicas, como

por ejemplo: soporte por defecto de bucles paralelos y serialización de accesos concurrentes de un conjunto de variables, sincronización declarativa basada en enlaces en lugar de hilos, manejadores especiales para deshacer invocaciones previas de servicios cuando se lanza un fallo y decidir si es necesario finalizar la ejecución de una parte del código, y elementos especializados en recibir mensajes de la red.

Por otro lado, WS-BPEL carece de algunas características comunes en otros lenguajes, como funciones, sentencias *switch* de valores enteros, vectores u orientación a objetos, entre otras. Esto se debe a que el propósito de los lenguajes estudiados es muy diferente: WS-BPEL se utiliza como un lenguaje de ejecución de procesos de negocio; C, Fortran y Ada como lenguajes de programación estructurada; C++, C#, ASP.NET y Java como lenguajes de programación orientada a objetos; SQL como un lenguaje de consulta, y XSLT como un lenguaje de transformación de documentos XML.

Este estudio concluye que puede mejorarse el conjunto de operadores de mutación para WS-BPEL. Se han incluido cuatro operadores de mutación de expresiones: dos que insertan código (EAI y EUI) y dos que modelan errores que pueden cometer los programadores al escribir a mano las expresiones XPath (ENC y ECI).

En el futuro, se pretende ampliar el conjunto de operadores para WS-BPEL con operadores que muten otras características de las expresiones XPath y con los operadores necesarios para medir algunos criterios de cobertura de código, como la cobertura de sentencia o de rama.

Referencias

[1] Dereziňska, A., "An experimental case study to applying mutation analysis for SQL queries". En: Ganzha, M. y Paprzycki, M. (Eds.), *Proceedings of the International Multiconference on Computer Science and Information Technology*. *Mrągowo (Polonia), 12-14 de octubre de 2009*, pp. 559-566, 2009.

[2] Alves, A. et al., Eds., "Web Service Business Process Execution Language (WS-BPEL) 2.0". Organization for the Advancement of Structured Information Standards (OASIS), 2007.

[3] Estero-Botaro, A., Palomo-Lozano, F. y Medina-Bulo, I., "Mutation Operators for WS-BPEL 2.0". En: *Proceedings of XXI International Conference on Software & Systems Engineering and their Applications*. *Paris (Francia), 9-11 de diciembre de 2008*, 2008.

[4] Estero-Botaro, A., Palomo-Lozano, F. y Medina-Bulo, I., "Quantitative Evaluation of Mutation Operators for WS-BPEL Compositions". En: Bilof, R. (Ed.), *Proceedings of III International Conference on Software Testing, Verification, and Validation Workshops*. *Paris (Francia), 6-10 de abril de 2010*, pp. 142-150, 2010.

[5] Jia, Y. y Harman, M., *An Analysis and Survey of the Development of Mutation Testing*. Informe técnico, King's College, London, Reino Unido, 2009.

[6] Agrawal, H. et al., *Design of Mutant Operators for the C Programming Language*. Informe técnico, Purdue University, West Lafayette, Indiana, 1989.

[7] Delamaro, M. E. y Maldonado, J. C., "Proteum - A Tool for the Assessment of Test Adequacy for C Programs". En: *Proceedings of the Conference on Performability in Computing System*. *East Brunswick (Nueva Yersey)*, *julio de 1996*, pp. 79-95, 1996.

[8] Barbosa, E. F., Maldonado, J. C. y Vincenzi, A. M. R., "Toward the determination of sufficient mutant operators for C", *Software Testing, Verification and Reliability*, vol. 11, nº 2, pp. 113-136, 2001.

[9] King, K. N. y Offutt, A. J., "A FORTRAN Language System for Mutation-based Software Testing", *Software - Practice and Experience*, vol. 21, nº 7, pp. 685-718, 1991.

[10] Offutt, A. J., Lee, A., Rothermel, G., Untch, R. H. y Zapf, C., "An experimental determination of sufficient mutant operators", *ACM Transactions on Software Engineering and Methodology*, vol. 5, pp. 99-118, 1996.

[11] Offutt, A. J., Voas, J. y Payne, J., *Mutation Operators for Ada*. Informe técnico, George Mason University, Fairfax, Virginia, 1996.

[12] Zhang, H., "Mutation Operators for C++", 2003. (Web). En: http://people.cis.ksu.edu/~hzh8888/mse_project.htm. Último acceso: 28 de febrero de 2011.

[13] Dereziński, A., "Quality Assessment of Mutation Operators Dedicated for C# Programs". En: Kellenberger, P. (Ed.), *Proceedings of VI International Conference on Quality Software*. *Beijing (China), 27-28 de octubre de 2006*, pp. 227-234, 2006.

[14] Dereziński, A., "Advanced mutation operators applicable in C# programs". En: Krzystof, S. (Ed.), *Software Engineering Techniques: Design for Quality*, Springer, 2007.

[15] Derezinska, A. y Szustek, A., "Tool-Supported Advanced Mutation Approach for Verification of C# Programs". En: Werner, B. (Ed.), *Proceedings of III International Conference on Dependability of Computer Systems*. *Szklarska Poreba (Polonia), 26-28 de junio de 2008*, pp. 261-268, 2008.

[16] Mansour, N. y Houri, M., "Testing web applications", *Information and Software Technology*, vol. 48, nº 1, pp. 31-42, 2006.

[17] Ma, Y. S., Kwon, Y. R. y Offutt, J., "Inter-Class Mutation Operators for Java". En: Kawada, S. (Ed.), *Proceedings of XIII International Symposium on Software Reliability Engineering*. *Annapolis (Maryland), 12-15 de noviembre de 2002*, pp. 352-363, 2002.

[18] Offutt, J., Ma, Y. S. y Kwon, Y. R., "The class-level mutants of MuJava". En: Anderson, K. (Ed.), *Proceedings of the 2006 International Workshop on Automation of Software Test*. *Shanghai (China), 20-28 de mayo de 2006*, pp. 78-84, 2006.

[19] Ma, Y. S., Offutt, J. y Kwon, Y. R., "MuJava: a mutation system for Java". En: Anderson, K. (Ed.), *Proceedings of XXVIII International Conference on Software Engineering*. *Shanghai (China), 20-28 de mayo de 2006*, pp. 827-830, 2006.

[20] Ma, Y. S., Offutt, J. y Kwon, Y., "MuJava: An Automated Class Mutation System", *Software Testing, Verification and Reliability*, vol. 15, nº 2, pp. 97-133, 2005.

[21] Ma, Y. S., Kwon, Y. R. y Kim, S. W., "Statistical Investigation on Class Mutation Operators", *ETRI Journal*, vol. 31, nº 2, pp. 140-150, 2009.

[22] Bradbury, J. S., Cordy, J. R. y Dingel, J., "Mutation Operators for Concurrent Java (J2SE 5.0)". En: *Proceedings of II Workshop on Mutation Analysis*. *Raleigh (Carolina del Norte), 7-10 de noviembre de 2006*, pp. 83-92, 2006.

[23] Ji, C., Chen, Z., Xu, B. y Wang, Z., "A New Mutation Analysis Method for Testing Java Exception Handling". En: Kellenberger, P. (Ed.), *Proceedings of XXXIII Annual IEEE International Computer Software and Applications Conference*. *Seattle (Washington), 20-24 de julio de 2009*, pp. 556-561, 2009.

[24] Madeyski, L. y Radyk, N., "Judy - a mutation testing tool for Java", *IET Software*, vol. 4, nº 1, pp. 32-42, 2010.

[25] Tuya, J., Suárez-Cabal, M. J. y de la Riva, C., "Mutating database queries", *Information and Software Technology*, vol. 49, nº 4, pp. 398-417, 2007.

[26] Tuya, J., Suárez-Cabal, M. J. y de la Riva, C., "SQLMutation: A tool to generate mutants of SQL database queries". En: *Proceedings of II Workshop on Mutation Analysis*. *Raleigh (Carolina del Norte), 7-10 de noviembre de 2006*, 2006.

[27] Lonetti, F. y Marchetti, E., "X-MuT: A Tool for the Generation of XSLT Mutants". En: Werner, B. (Ed.), *Proceedings of VII International Conference on the Quality of Information and Communications Technology*. *Porto (Portugal), 29 de septiembre - 2 de octubre de 2010*, pp. 280-285, 2010.

[28] Kim, S., Clark, J. A. y McDermid, J. A., "Class Mutation: Mutation Testing for Object-Oriented Programs". En: *Proceedings of Conference on Object-Oriented Software Systems*. *Erfurt (Alemania), octubre de 2000*, pp. 9-12, 2000.

Reseña sobre el taller de Pruebas en Ingeniería del Software 2010 (PRIS)

Claudio de la Rova
Departamento de Informática
Universidad de Oviedo
claudio@uniovi.es

El V Taller sobre Pruebas en Ingeniería del Software (PRIS 2010: http://in2test.lsi.uniovi.es/pris2010/) se celebró en Valencia el 7 de Septiembre de 2010, en el marco de las XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2010) y del III Congreso Español de Informática (CEDI 2010). Este taller se configura como un foro de discusión de las diferentes actividades relacionadas con la prueba del software, tanto a nivel industrial como académico y de formación.

En esta quinta edición del taller se presentaron un total de 8 contribuciones seleccionadas tras haber sido sometidas a un proceso de revisión por pares y dos charlas invitadas: "FITTEST – Testeo del Internet del Futuro" impartida por Oscar Pastor, director del Centro de Investigación en Métodos de Producción Software (ProS) de la Universidad Politécnica de Valencia y "Business Driven Test Management – Una buena práctica en la gestión de las pruebas" impartida por E. van Driel, responsable de la unidad de consultoría de pruebas de Sogeti España.

Las contribuciones presentaron temáticas variadas. En un primer bloque se presentaron trabajos y estudios de orientación fundamentalmente investigadora relacionados con Internet y las Arquitecturas Orientadas a Servicios (SOAs). Dos de ellos están relacionados con revisiones sistemáticas en la pruebas en SOAs: "Revisión Sistemática: Pruebas del Software para SOA con UML" por M.P. Romay, L. Fernández-Sanz y R. Tejedor y "Protocolo para la Revisión Sistemática de Estudios sobre Pruebas en SOAs con Enlace Dinámico" por M. Palacios. J. García-Fanjul y J. Tuya. Un tercer trabajo, "Equivalencia entre los Operadores de Mutación definidos para WS-BPEL 2.0 y los definidos para otros lenguajes" por J. Boubeta, I. Medina y A. García, presenta un estudio que compara los operadores de mutación definidos en el lenguaje BPEL con respecto a otros definidos en otros entornos y lenguajes más conocidos.

En un segundo bloque de contribuciones, se expusieron también trabajos de investigación en aspectos muy variados de las pruebsa del software. "Propuesta de Optimización en la Prueba de Mutaciones en Java" por I. Medina, L. Gutiérrez, J. J. Domínguez, describe la aplicación de una técnica de mutación evolutiva con el objetivo de reducir el número de mutantes que se generan para la prueba de código en Java. "Un Portal Web para Investigación y Docencia en Testing Combinatorio" por B. Pérez y M. Polo, presenta una herramienta con orientación a la docencia e investigación que implementa diferentes algoritmos basados en pruebas combinatorias. "Oráculo de Prueba Automatizado para Programas de Procesamiento en XML" por D.S. Kim-Park, C. de la Riva y J. Tuya, describe la implementación de un oráculo de pruebas para la evaluación de las salidas en programas basados en XML. "Verificación de Datos en la GUI como un Aspecto Separado de las Aplicaciones" por P.L. Mateo, G. Martínez y D. Sevilla, presenta un framework que describe una capa de verificación entre la interfaz gráfica de usuario y la lógica de aplicación con el objetivo de tratar la prueba de los datos de entrada de usuario de forma independiente al resto de la aplicación.

En el último bloque de presentaciones, el trabajo "Pruebas Funcionales Evolutivas en la Industria" pro T. Vos, A. Vaas y B. Marín, describe la aplicación de técnicas evolutivas para la prueba del software, bien conocidas en el ámbito académico, en dos casos de estudio industriales dentro del proyecto europeo EvoTest.

Finalmente, deseo agradecer al resto del comité organizador del taller, a los organizadores de las conferencias que albergaron este taller, a los participantes y en especial al editor de REICIS por el soporte e interés mostrado en este taller. Todos ellos contribuyen tanto a la consolidación de diversos grupos de investigación en pruebas como a la aportación de una visión práctica que pueda mejorar finalmente la formación y los procesos de pruebas realizados en la práctica industrial.

REICIS

ACTUALIDAD INVITADA

Principales actividades de IFIP previstas para los próximos años

Ramón Puigjaner Trepat
International Federation for Information Processing

Introducción

El propósito de esta breve reseña es proporcionar información sobre las actividades globales de IFP previstas para los próximos años. En la misma, no incluyo las actividades específicas organizadas por los grupos de trabajo (WG) y los comités técnicos (TC) que sería muy prolijo poder presentar en un espacio breve.

World CIO Forum

El World CIO Forum preparado por Mr. Forrest Lin (Secretario adjunto del Chinese Institute of Electronics) está previsto que tenga lugar en Shenzhen (China) del 1 al 4 de noviembre de 2011. Se trata de una nueva actividad que tiene por objeto reunir a los *Chief Information Officers* (CIO) con objeto de poner en común experiencias, problemas de su actividad y debatir la forma de atacarlos y resolverlos. Para información sobre esta actividad puede consultarse su página web (www.worldcioforum.com).

En función del éxito que se logre en esta edición está previsto darle continuidad con periodicidad anual o bienal para lo cual, oportunamente, se convocará a las sociedades miembro de IFIP para que se postulen como organizadores de próximas ediciones.

WITFOR

Este serie de eventos tiene como objetivo poner en contacto los informáticos del hemisferio norte (países mayoritariamente desarrollados) con los del sur (países mayoritariamente en vías de desarrollo o emergentes) tanto para discutir a nivel político cómo la informática puede ayudar al progreso de esos países como para poner en común las experiencias de aplicación de la informática y la telecomunicaciones para este objetivo de progreso.

Después de las ediciones anteriores de Vilnius (Lituania) en 2003, Gaborone (Botswana) en 2005, Adis Abeba (Etiopía) en 2007 y Hanoi (Vietnam) en 2009, la edición del World Information Technology Forum WITFOR prevista para este año ha sido necesaria retrasarla hasta 2012 al haber fracasado la búsqueda de un país interesado en organizarlo. Finalmente se celebrará en Delhi (India) en abril de 2012. Las novedades organizativas que presenta esta edición son:

- La creación de un comité de programa con la misión de coordinar las presentaciones y evitar la duplicidades en los temas tratados en la distintas comisiones temáticas
- La reducción del número de temas a tratar (de 8 a 10 en las ediciones anteriores) a sólo cuatro, los de mayor interés para el país organizador

Después de haber salido de Europa y recorrido África y Asia, IFIP pretende que las próximas ediciones de WITFOR tengan lugar en América Latina y, además, que recupere su secuencia de realización en años impares. Así para 2013 se han establecido contactos esperanzadores para organizarlo en Paraguay. Y para el 2015 se ascendería hacia el norte, hasta América Central, para organizarlo probablemente en Panamá o Costa Rica.

WCC

El World Computer Congress (WCC), tal vez el evento más emblemático en la historia de IFIP, ha evolucionado en los 50 años de su vida. Ha pasado de ser un congreso generalista en los años sesenta, setenta y ochenta del siglo XX, cuando era un lugar de encuentro entre profesionales informáticos del Bloque del Este y del Oeste, a un conjunto de conferencias en paralelo en las décadas de los noventa y del 2000. Un buen ejemplo de ellos es su última edición celebrada en Brisbane (Australia) en septiembre de 2010. Sin embargo, en cada una de las ediciones más recientes se ha ido viendo que estas estructuras estaban agotadas y que era necesario buscar otras nuevas.

Así, para 2012 se pretende explorar un tema de actualidad y tratarlo en profundidad. Para ese año se ha escogido el de "Putting the IT professional first: Towards a secure, reliable and innovative information society". Los objetivos previstos para el WCC 2012 se centran en el profesional de TIC de hoy en día y de mañana, actor fundamental de la sociedad de la información y el congreso se focalizará en tres temas esenciales:

- Tecnología: Nuevas formas de trabajo colaborativo e innovación
- Perfiles y ética: Aspectos éticos de los profesionales y de los negocios TIC.
- Prevención: Prevención de los crímenes cibernéticos y protección de las infraestructuras críticas.

Está previsto que dicha edición se celebre en Ámsterdam (Países Bajos) en septiembre de 2012. En función de los resultados que se obtengan en este experimento, se mantendrá la fórmula o se explorarán nuevas alternativas.

WCCE

El World Computer Congress on Education (WCCE), dedicado tanto a la enseñanza de la informática como a la aplicación de la informática en la enseñanza, sigue con su periodicidad trienal. Está previsto que el próximo congreso se celebre en Polonia en 2013. Los detalles de este evento todavía están por fijar.

Perfil profesional

Ramon Puigjaner Trepat es catedrático de Arquitectura y Tecnología de Computadores de la Universitat de les Illes Balears desde 1988 y Emérito desde 2010. Autor de un libro y de cerca de 200 trabajos de investigación sobre evaluación de prestaciones de sistemas informáticos y de comunicaciones ha sido director de numerosos proyectos de investigación españoles e internacionales y de cooperación internacional. Es Premio de Honor del CLEI por su colaboración con Latinoamérica en 2007 así como Premio Nacional de Informática por su trayectoria profesional en 2007. Es Doctor Honoris Causa por la Universidad Nacional de Asunción y vicepresidente de la Federación Internacional de Informática IFIP (International Federation for Information Processing).