XV Jornadas de Ingeniería del Software y Bases de Datos
JISBD 2006
José Riquelme - Pere Botella (Eds)

MODELOS Y ALGORITMOS PARA LA GENERACIÓN DE OBJETIVOS DE PRUEBA

Javier J. Gutiérrez, María J. Escalona, Manuel Mejías y Jesús Torres

Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla
Avd. Reina Mercedes sn. 41012. España
{javierj, escalona, risoto, jtorres}@lsi.us.es

Palabras clave: Modelos de prueba, Algoritmos de prueba, Pruebas del sistema.

Resumen. *Una tarea vital en el desarrollo del software es probar la correcta implementación de los requisitos funcionales. Los casos de uso se utilizan con mucha frecuencia para definir la funcionalidad de un sistema software en las etapas tempranas de su desarrollo. Este trabajo expone la falta de automatización en las propuestas existentes para la generación de casos de prueba y presenta una nueva propuesta para obtener de manera sistemática objetivos de prueba a partir de los casos de uso del propio sistema.*

1. INTRODUCCIÓN

El aumento de la complejidad de los sistemas software propicia el aumento de la necesidad de garantizar su calidad. Las pruebas del sistema son una técnica que ayuda a garantizar la calidad del software. Las pruebas del sistema son un procedimiento de caja negra para verificar la satisfacción de los requisitos del sistema a prueba (SUT en inglés) [3].

El posible desarrollar varios tipos de pruebas del sistema, como pruebas funcionales, de usabilidad, se seguridad, navegación, etc. Este trabajo está centrado en las pruebas funcionales desde el punto de vista de los actores del sistema y, especialmente, de actores humanos que interactúan a través de interfaces gráficas. Así, un caso de prueba sustituye a un actor y simula su comportamiento para verificar que el sistema hace lo que se espera de él. Por esta razón, el principal artefacto para obtener pruebas del sistema son los requisitos funcionales del SUT, ya que ellos describen todo el comportamiento esperado que debe ser probado [12].

Nuestra propuesta especifica el comportamiento del sistema mediante casos de uso. Los casos de uso ofrecen una visión general del sistema y son fáciles de estudiar y validar por parte de

usuarios no expertos en ingeniería del software. Es también muy común en la industria del software identificar casos de prueba a partir de los casos de uso [4]. En fases tempranas del desarrollo, cuando los requisitos están siendo descubiertos, definidos y negociados, es mucho más sencillo modificar los casos de uso definidos como texto libre o texto estructurado que requisitos formales.

Este trabajo propone un proceso sistemático para generar objetivos de prueba a partir de los casos de uso. Los objetivos de prueba obtenidos a partir de los casos de uso definen qué debe ser probado para asegurar la correcta y completa implementación de los casos de uso [19]. El diseño de buenos casos de uso es una tarea vital para el proceso de pruebas, además de para el proceso de desarrollo. Sin embargo, el perfil de pruebas de UML (UMLTP [19]) no define ninguna notación para representar objetivos de prueba. Nosotros hemos resuelto esta carencia mediante el uso de diagramas de actividades y secuencias de actividades.

En un trabajo previo expusimos cómo generar casos de prueba a partir de los casos de uso utilizando algunas de las propuestas existentes [9]. Sin embargo, dichas propuestas están muy basadas en el trabajo manual y en las decisiones de los ingenieros de pruebas. La generación automática de los objetivos de prueba presentada en este trabajo continúa esta línea de investigación y es también el primer paso para obtener un proceso completo y sistemático para la generación de casos de prueba.

La organización de este trabajo se describe a continuación. En la sección 2 se describe brevemente el estado del arte en cuanto a propuestas de generación de pruebas. En la sección 3 se describe el modelo de plantilla utilizado para la definición de casos de uso. En la sección 4 se presenta el caso práctico utilizado como ejemplo a lo largo de este trabajo. En la sección 5 se detalla cómo obtener objetivos de prueba a partir de los casos de uso. Finalmente, en la sección 6 se exponen las conclusiones y los trabajos futuros.

2. ESTADO DEL ARTE

Existen varias propuestas para la generación de objetivos de prueba. Dos estudios que analizan estas propuestas son [5] y [10]. En total, ambos estudios analizan 21 propuestas. Los siguientes párrafos describen algunas de estas propuestas y sus principales puntos débiles.

Las propuestas existentes pueden clasificarse en tres grupos dependiendo de los artefactos usados para la generación de objetivos de prueba. El primer grupo engloba las propuestas que generan objetivos de prueba directamente a partir de los casos de uso, como [2] y [11]. El segundo grupo engloba las propuestas que generan un modelo de comportamiento a partir de los casos de uso y genera casos de prueba a partir de él, como [18], [13] o [17]. Algunas de las notaciones utilizadas para el modelo de comportamiento son: diagramas de actividades, diagramas de estados o árboles de escenario. El tercer grupo engloba propuestas centrabas en variables operacionales y valores de prueba. Estas propuestas identifican las variables de un caso de uso y realizan particiones de los dominios de dichas variables, como [1] y [16]. Las propuestas basadas directamente en los casos de uso escritos en prosa son difíciles de sistematizar y automatizar. Tampoco hemos encontrado ninguna propuesta sistemática para identificar variables, dominios y particiones. Las

propuestas basadas en modelos de comportamiento están descritas con poco detalle y no ofrecer herramientas de soporte.
Las propuestas existentes también pueden clasificarse en dos grupos dependiendo del alcance de los objetivos generados. El primer grupo engloba las propuestas que generan objetivos de prueba a partir de casos de uso de manera aislada, como [2], [1] o [17]. El segundo grupo engloba las propuestas que generan objetivos de prueba a partir de secuencias de casos de uso, como [18] o [13]. Sin embargo, ninguna de las propuestas del primer grupo permite obtener de manera automática modelos de comportamiento ni objetivos de prueba a partir de casos de uso.
Los párrafos anteriores justifican la necesidad de una nueva propuesta para generar un modelo de comportamiento y obtener casos de prueba de manera automática.

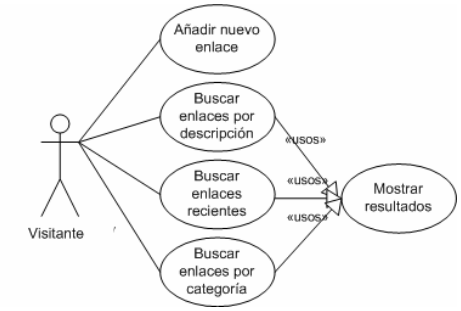
3. UN MODELO DE CASO DE USO Y PLANTILLA PARA PRUEBAS

Una técnica ampliamente utilizada en la industria para definir casos de uso son las plantillas. Una plantilla combina la prosa con una estructura concreta. La generación sistemática de objetivos de prueba implica una serie de restricciones. Una de ellas es la necesidad de definir un modelo concreto de plantillas de casos de uso que pueda ser manipulado de manera sistemática y automática sin perder la ventaja de usar texto en prosa.
Como modelo hemos utilizado el modelo de requisitos propuesto en la metodología Navigational Development Technique (NDT) [6], la cual ofrece un modelo de requisitos completo, formal y flexible. NDT propone una plantilla tabular para definir requisitos funcionales mediante casos de uso. Hemos elegido el modelo de requisitos y las plantillas de NDT por dos motivos principales. En primer lugar, este modelo está basado en un metamodelo formal definido mediante UML [6], [14]. En segundo lugar, el modelo de requisitos de NDT ha sido aplicado en muchos proyectos reales y complejos, como [7], [8] y [21].
Hemos desarrollado una pequeña extensión a la plantilla de NDT para mejorar la generación de pruebas. Un ejemplo de la plantilla extendida se muestra en el caso de uso definido en la figura 1.
Varios campos de la plantilla, como las precondiciones, post-condiciones, etc. Son comunes a otras técnicas de requisitos y se han descrito en numerosos trabajos, como [4] y [6], por lo que su definición se ha omitido. Los siguientes párrafos describen el resto de los campos de la plantilla.
Secuencia principal: La secuencia principal está compuesta de los pasos del caso de uso que permiten al actor obtener su objetivo. Cada paso de la secuencia principal está compuesto de un identificador (los pasos se numeran consecutivamente), quién realiza el caso de uso (un actor o el sistema) y la acción realizada.
Errores / alternativas: Estos pasos describen el comportamiento del sistema cuando ocurre un error o se ejecuta un flujo alternativo en el caso de uso. Estos pasos tienen los mismos elementos que los pasos de la secuencia principal y algunos elementos adicionales. El identificador está compuesto de dos números, el primero debe hacer referencia a un paso

de la secuencia principal. El segundo número permite distinguir entre las diferentes alternativas o errores de un paso de la secuencia principal. Cada paso de error o de alternativa debe incluir una condición para indicar cuando se ejecutará el paso. También deben incluir el resultado del paso, por ejemplo, finalizar el caso de uso, continuar el caso de uso o repetir un conjunto de pasos de la secuencia principal.
Las condiciones de un error o alternativa se clasifican en precondiciones e invariantes. Las precondiciones se evalúan antes de que el paso de la secuencia principal comience. Las invariantes se evalúan durante la ejecución del paso de la secuencia principal.
Resultado: El resultado indica qué pasos de la secuencia principal o de la alternativa terminan el caso de uso y cuál es el resultado obtenido por el actor o actores participantes. Algunos casos de uso no tienen un resultado visible o la definición del resultado está fuera del alcance del caso de uso. En ese caso, se indica que el resultado no es visible.

4. CASO PRÁCTICO

El sistema a prueba (SUT) es una aplicación web que permite gestionar un catálogo de enlaces "on-line" (descargada de www.codecharge.com). El SUT contempla dos actores, el actor visitante y el administrador. Sin embargo, en este caso de estudio nos centraremos únicamente en el actor visitante. El diagrama de casos de uso para este actor se muestra en la figura 1.



Nombre	UC-02. Buscar enlaces por descripción
Precondición	No.
Secuencia Principal	**1.** El visitante solicita al sistema buscar enlaces por su descripción. **2.** El sistema solicita la descripción. **3.** El visitante introduce la descripción. **4.** El sistema busca los enlaces cuya descripción case con la introducida por el visitante. **5.** El sistema muestra los resultados.
Errores / alternativas	**3.1.i.** En cualquier momento, el visitante puede cancelar la búsqueda, entonces el caso de uso termina. **4.1.p.** Si el visitante introduce una descripción vacía, entonces el sistema busca todos los enlaces almacenados y el resultado es continuar la ejecución del caso de uso. **4.2.i.** Si el sistema encuentra un error realizando la búsqueda, entonces muestra un mensaje de error y este caso de uso termina. **5.1.p.** Si el resultado de la búsqueda está vacío, entonces el sistema muestra un mensaje y este caso de uso termina.
Resultados	**5.** El sistema muestra los resultados de la búsqueda. **3.1.i.** Fuera de los límites de este caso de uso. **4.2.i.** Mensaje de error. **5.1.p.** Mensaje de no resultados.
Post conditiones	No

Figura 1. Diagrama de casos de uso y caso de uso.

El caso de uso que usaremos en el caso práctico es “Buscar enlace por descripción”. No entraremos en detalles sobre cómo se muestran los enlaces encontrados. La definición de este caso de uso se muestra en la plantilla de la figura 1. En dicha plantilla las precondiciones se indican con una letra ‘p’ y las invariantes con una letra ‘i’. Los componentes de la plantilla que no son relevantes en este caso práctico han sido omitidos.

5. OBJETIVOS DE PRUEBA A PARTIR DE CASOS DE USO

Para obtener objetivos de prueba, en primer lugar se construye un modelo de comportamiento a partir de un caso de uso. En segundo lugar, dicho modelo se recorre para identificar los objetivos de prueba.

5.1. Construcción del modelo de comportamiento

La primera tarea es construir el modelo de comportamiento a partir del caso de uso. Como se mencionó en la sección 1, utilizamos un diagrama de actividades para representar el comportamiento del caso de uso. Este modelo representa los distintos escenarios o instancias de un caso de uso. Los siguientes párrafos describen los pasos para generar un modelo de comportamiento a partir de un caso de uso definido mediante la plantilla descrita en la sección 3

Cada paso de la secuencia principal es una actividad del modelo de comportamiento. Las transiciones se añaden entre las actividades consecutivas. Un modelo de comportamiento tiene un único punto de origen, el cuál es la primera actividad del caso de uso. Un modelo de comportamiento tiene tantos puntos de terminación como resultados. Cada paso de error o alternativa es una decisión en el diagrama de actividades. Dicha decisión evalúa la condición de la alternativa. Si hay una secuencia de decisiones que pertenecen al mismo actor o al sistema, pueden unirse en una sola decisión. Si el paso alternativo o erróneo incluye alguna acción, re presenta como una nueva actividad.

Finalmente, las actividades se agrupan mediante clasificadores. Cada clasificador contiene todas las actividades y decisiones realizadas por un actor o por el sistema. La figura 2 muestra el modelo de comportamiento obtenido a partir del caso de uso de la figura 1.

El algoritmo “BuildBehaviourModel” describe el proceso de generación de los casos de prueba. Las funciones auxiliares tienen un nombre descriptivo por lo que su definición ha sido omitida.

```
algorithm BuildBehaviourModel
var
        model : ACTIVITYGRAPH
        alternativeSteps : LIST[USECASESTEP]
        step : USECASESTEP
init
foreach (step in useMase.mainSequence)
        alternativeSteps = useCase.getAlternatives(pre, step)
        if ( not_empty(alternativeSteps) )
                traverse_alternativeSteps(behaviourModel, alternativeSteps)
```

```
        end if
        behaviourModel.addActivity(step)
        alternativeSteps = useCase.getAlternatives(inv, step)
        if ( not_empty(alternativeSteps) )
                traverse_alternativeSteps(behaviourModel, alternativeSteps)
        end if
end foreach

function traverse_alternativeSteps(behaviourModel, alternativeSteps)
        alternative : STEP
        decision = behaviourModel.addDesicion(alternativeSteps))
        foreach ( alternative in alsternativeSteps)
                if ( is_activity(alternative.action) )
                        node = behaviourModel.addActivity(alternative.action)
                end if
                if ( is_end(alternative.action) )
                        node = behaviourModel.addActivity(activityEnd)
                end if
                if ( is_gotoActivity(alternative.action) )
                        node = behaviourModel.getActivity(alternative.action)
                end if
                behaviouralModel.addTransition(decision, node)
        end foreach
end function
```

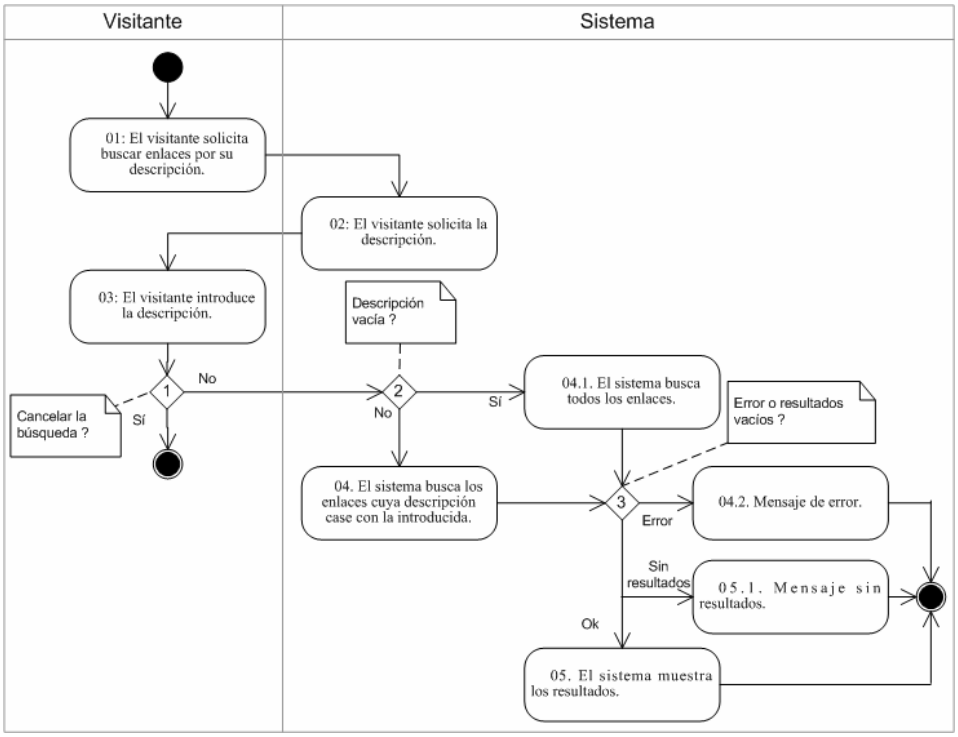


Figura 2. Modelo de comportamiento

Si cada paso del caso de uso incluye una única actividad, entonces el número máximo de nodos (actividades y decisiones) de un modelo de comportamiento es el número de pasos en la secuencia principal y el número de los pasos alternativos multiplicado por 2. Para el caso de uso de la figura 1, el máximo número de nodos es 5 + (4 x 2) = 13. El modelo de comportamiento de la figura 2 contiene sólo 11 nodos ya que el paso 3.1.i no genera ninguna actividad y las condiciones de los pasos 4.2.i y 5.1.p se han unido en la decisión 3.

5.2. Generación de objetivos de prueba

Después de la construcción del modelo de comportamiento, los objetivos de prueba se obtienen de manera sistemática. Como se ha mencionado, un objetivo de prueba es un conjunto de interacciones entre el sistema y un caso de prueba (que reemplaza a un actor) para verificar que el comportamiento del sistema es el comportamiento definido en sus casos de uso. Los objetivos de pruebas se definen como caminos a través del modelo de comportamiento o como diagramas de actividades dónde sólo existe un único camino.
Es posible utilizar varios criterios de cobertura para generar secuencias a partir de un diagrama de actividades. Por ejemplo, dos criterios clásicos son los criterios de todos los nodos y de todas las transiciones. En este caso, el criterio de selección escogido es el criterio de todos los escenarios (AS). Un conjunto de objetivos de prueba satisface el criterio AS si cada posible escenario del caso de uso es probado por un y solo un objetivo de prueba. Un escenario es una instancia de un caso de uso. El criterio AS asegura que todos los objetivos obtenidos son alcanzables y que ninguno de los objetivos está repetido. Dos objetivos de prueba son los mismos cuando tienen las mismas actividades en el mismo orden. El algoritmo “BuildTestObjectives” describe cómo generar los objetivos de prueba. Al igual que en el algoritmo “BuildBehaviourModel”, las funciones auxiliares tienen un nombre descriptivo y su definición ha sido omitida

```
algorithm BuildTestObjectives
var     objective : PATH
        objectives : LIST(PATH)
init
        objective = < empty >
        objectives = < empty >
        traverse(initialNode, path)
function traverse(in node, inout objective)
        if ( is_desicion(node) )
                traverse_desicion(node, objective)
                exit function
        end if
        objective.add(node)
        if ( isEnd(node) )
                objectives.add(objective)
                exit function
        end if
        nextNode = next_node(node)
```

```
        traverse(nextNode, objective)
end function
function traverse_desicion(in node, inout objective)
        foreach (alternative in node.alternatives)
                path.add(alternative)
                nextNode = next_node(alternative)
                traverse(nextNode, objective)
        end foreach
end function
```

Los objetivos de prueba obtenidos para el modelo de comportamiento de la figura 2 se muestran en la tabla 1. El primer objetivo de prueba es el objetivo de prueba de la secuencia principal del caso de uso.

Id	Caminos / Objetivos de prueba
1	01 -> 02 -> 03 -> D1(No) -> D2(No)-> 04 -> D3(Sin error & Resultados) -> 05
2	01 -> 02 -> 03 -> D1(No) -> D2(No)-> 04 -> D3(Sin error & Sin resultados) -> 05.1
3	01 -> 02 -> 03 -> D1(No) -> D2(No)-> 04 -> D3(Error) -> 04.2
4	01 -> 02 -> 03 -> D1(No) -> D2(Sí)-> 04.1 -> D3(Sin error & Resultados) -> 05
5	01 -> 02 -> 03 -> D1(No) -> D2(Sí)-> 04.1 -> D3(Sin error & Sin Resultados) -> 05.1
6	01 -> 02 -> 03 -> D1(No) -> D2(Sí)-> 04.1 -> D3(Error) -> 04.2
7	01 -> 02 -> 03 -> D1(Sí)

Tabla 1. Objetivos de prueba.

Como se ha mencionado, los objetivos de prueba también pueden expresarse como diagramas de actividades. La figura 3 muestra los diagramas de actividades correspondientes al camino 1 (3a) y al camino 7 (3b) de la tabla 1.

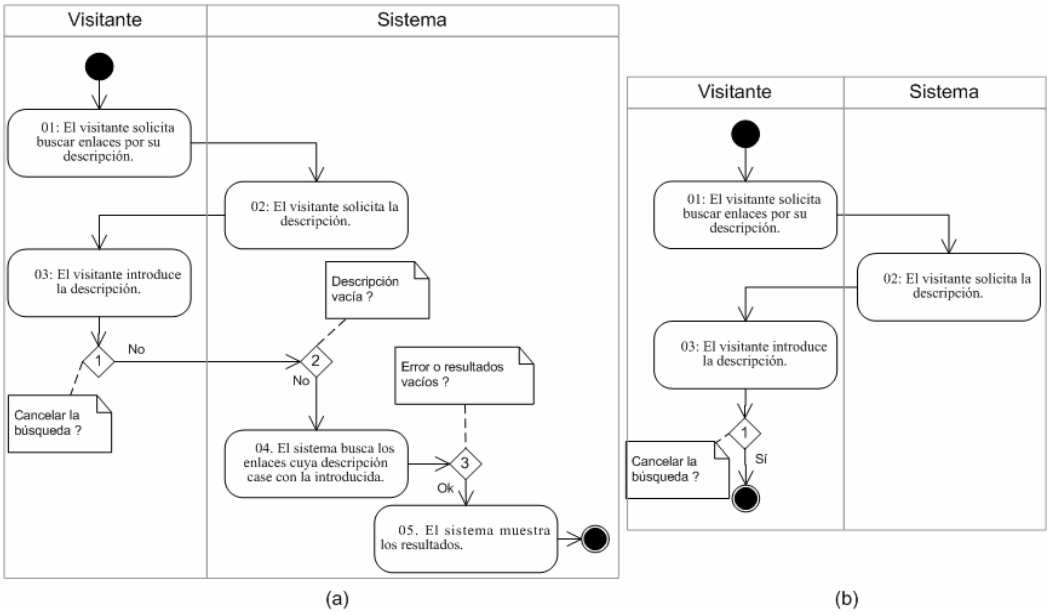


Figura 3. Objetivos de prueba representados con diagramas de actividades.

5.3. Cobertura de los casos de uso

La cobertura de los objetivos de prueba mide el número de escenarios de un caso de uso que cuenta con un caso de prueba. Los objetivos de la tabla 1 cubren el 100% de los escenarios del caso de uso.

$$\frac{\text{Objetivos de prueba}}{\text{Escenarios}} = \text{Cobertura}$$

Prioridad	Cobertura
0	No se genera ningún objetivo para el caso de uso.
1	Solo se genera un objetivo para la secuencia principal.
2	Se generan objetivos para la secuencia principal y todas las alternativas de los actores.
3	Se generan todos los objetivos posibles.

Figura 4. Medida de la cobertura de los objetivos de prueba.

La cobertura de los casos de uso por parte de los objetivos de prueba puede calcularse a partir de la fórmula de la figura 4. Una mayor cobertura implica más objetivos de prueba y, por tanto más casos de prueba. Una cobertura de 1 (el 100%) significa que cualquier escenario posible tendrá un caso de prueba.

La cobertura y el número de objetivos de prueba generados pueden determinarse por la relevancia o la frecuencia del caso de uso. Ambos elementos están incluidos en la propuesta de plantilla de NDT y vista en la sección 3. Un criterio de cobertura basado en la prioridad del caso de uso se muestra en la figura 4.

6. CONCLUSIONES

Los objetivos de prueba son básicos para realizar un proceso de pruebas con éxito. Los objetivos de prueba indican que casos de prueba deben ser construidos para probar la implementación de los casos de uso. Este trabajo ha presentado una nueva propuesta para la generación sistemática de objetivos de prueba a partir de casos de uso. En la sección 2 se ha justificado la necesidad de este nueva propuesta debido a que no hemos encontrado ningún otro trabajo que indique como derivar de manera automática objetivos de prueba a partir de casos de uso.

Nuestra propuesta soluciona esas carencias y genera los mismos objetivos de prueba que los trabajos citados en la sección 2. Sin embargo, los principales beneficios de nuestro trabajo son la presentación de un modelo semiformal para la definición de casos de uso y la generación automática de objetivos de prueba. Todos los objetivos generados son alcanzables y únicos, como se ha visto en la sección 5.2. Los algoritmos descritos en la sección 5 han sido implementados en una herramienta de código abierto. Esta herramienta puede ser descargada libremente desde www.lsi.us.es/~javierj/ y está siendo mejorada añadiendo gestión de bucles y soporte para XMI.

Nuestros trabajos futuros se orientan en la extensión de nuestra propuesta. Nuestra meta final es la generación de casos de prueba de manera automática. Un trabajo preliminar en este sentido puede consultarse en [9].

REFERENCIAS

[1] Bertolino, A., Gnesi, S. 2004. PLUTO: A Test Methodology for Product Families. Lecture Notes in Computer Science. Springer-Verlag Heidelberg. 3014 / 2004. pp 181-197.
[2] Binder, R.V. 1999. Testing Object-Oriented Systems. Addison Wesley.
[3] Burnstein, I. 2003. Practical software Testing. Springer Professional Computing. USA.
[4] Cockburn, A. 2000. Writing Effective Use Cases. Addison-Wesley 1st edition. USA.
[5] Denger, C. Medina M. 2003. Test Case Derived from Requirement Specifications. Fraunhofer IESE Report.
[6] Escalona M.J. 2004. Models and Techniques for the Specification and Analysis of Navigation in Software Systems. Ph. European Thesis. University of Seville. Seville, Spain.
[7] Escalona M.J. Martín-Pradas A., De Juan L.F, Villadiego D., Gutiérrez J.J. 2005 El Sistema de Información de Autoridades del Patrimonio Histórico Andaluz. V Jornadas de Bibliotecas Digitales. Granada, Spain.
[8] Gutierrez J.J. Escalona M.J. Mejías M. Torres J. 2004. Aplicando técnicas de testing en sistemas para la difusión Patrimonial. TURITEC'2004. pp. 237-252. Málaga, Spain.
[9] Gutiérrez JJ, Escalona MJ, Mejías M, Torres J. 2005. A practical approach of Web System Testing. Advances in Information Systems Development: Bridging the gap between Academia and Industry. pp. 659-680. Ed. Springer Verlag Karlstad, Sweden.
[10] Gutiérrez, J.J., Escalona M.J., Mejías M., Torres, J. 2006. Generation of test cases from functional requirements. A survey. 4º Workshop on System Testing and Validation. Potsdam. Germany.
[11] Heumann , J. 2002. Generating Test Cases from Use Cases. Journal of Software Testing Professionals.
[12] Myers G. 2004. The art of software testing. Second edition. Addison-Wesley. USA.
[13] Nebut C. Fleury F. Le Traon Y. Jézéquel J. M. 2006. Automatic Test Generation: A Use Case Driven Approach. IEEE Transactions on Software Engineering Vol. 32. 3. March.
[14] Koch N. Zhang G. Escalona M. J. 2006. Model Transformations from Requirements to Web System Design. Webist 06. Portugal.
[15] Offutt, J. et-al. 2003. Generating Test Data from Sate-based Specifications. Software Testing, Verification and Reliability. 13, 25-53. USA.
[16] 16 T. J., Balcer M. J. 1988. Category-Partition Method. Communications of the ACM. 676-686.
[17] Ruder A. 2004. UML-based Test Generation and Execution. Rückblick Meeting. Berlin.
[18] Labiche Y., Briand, L.C. 2002. A UML-Based Approach to System Testing, Journal of Software and Systems Modelling (SoSyM) Vol. 1 No.1 pp. 10-42.
[19] Object Management Group. 2003. The UML Testing Profile. www.omg.org
[20] Object Management Group. 2003. Unified Modelling Language 2.0. www.omg.org
[21] Escalona M.J. Gutiérrez J.J. Villadiego D. León A. Torres A.H. 2006. Practical Experiences in Web Enginering. 15th International Conference On Information Systems Development. Budapest, Hungary, 31 August – 2 September