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


# MODELOS DE PRUEBAS PARA PRUEBAS DEL SISTEMA

**Javier J. Gutiérrez, María J. Escalona, Manuel Mejías y Antonia M. Reina**

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

**Palabras clave:** Pruebas del sistema, modelos de prueba, generación de pruebas.

**Resumen**. *Uno de los objetivos de la fase de pruebas del sistema es verificar que el comportamiento externo del sistema software satisface los requisitos establecidos por los clientes y futuros usuarios del mismo. A medida que aumenta la complejidad de los sistemas software y aumenta la demanda de calidad, se hacen necesarios procesos y métodos que permitan obtener buenos conjuntos de pruebas del sistema. Este trabajo describe los modelos necesarios para generar de manera sistemática un conjunto de pruebas que permitan verificar la implementación de los requisitos funcionales de un sistema software.*

## 1. INTRODUCCIÓN

La fase de pruebas del sistema tiene como objetivo verificar el sistema software para comprobar si este cumple sus requisitos. Dentro de esta fase pueden desarrollarse varios tipos distintos de pruebas en función de los objetivos de las mismas. Algunos tipos son pruebas funcionales, pruebas de usabilidad, pruebas de rendimiento, pruebas de seguridad, etc. Este trabajo se centra en pruebas funcionales de aplicaciones con interfaces gráficas. Estas pruebas verifican que el sistema software ofrece a los actores humanos la funcionalidad recogida en su especificación.

Una de las técnicas más empleadas para la especificación funcional de sistemas software son los casos de uso. Las principales ventajas de los casos de uso son que ocultan los detalles internos del sistema, son rápidos de construir, fáciles de modificar y entender por los clientes y futuros usuarios del sistema [2] y pueden aplicarse a distintos tipos de sistemas [3] y [5].

Actualmente, existe un amplio número de propuestas que describen cómo generar pruebas del sistema a partir de los casos de uso. Aunque la generación de pruebas se adapta a la filosofía propuesta por MDA, tal y como mostraremos a continuación, ninguna de estas propuestas define su proceso en base a las técnicas de MDA. Por este motivo, una de las principales

carencias es la falta de modelos que recojan la información necesaria en el proceso de generación de pruebas. Este trabajo se centra en la definición de un conjunto de modelos que den soporte al proceso de generación de pruebas del sistema desde una perspectiva MDA.
La estructura de este trabajo se define a continuación. En la sección 2 se definen las ideas generales de un proceso de generación de pruebas del sistema. En la sección 3 se detallan los distintos modelos necesarios para la generación de pruebas. En la sección 4 se definen la herramienta que hemos desarrollado. En la sección 5 se resumen otros trabajos relacionados. Finalmente, en la sección 6 se exponen las conclusiones y trabajos futuros.

## 2. EL PROCESO DE GENERACIÓN DE PRUEBAS DEL SISTEMA

Toda prueba consta tradicionalmente de tres elementos: interacciones entre el sistema y la prueba, valores de prueba y resultados esperados. Los dos primeros elementos permiten realizar la prueba y el tercer elemento permite evaluar si la prueba se superó con éxito o no. Un proceso de pruebas consta generalmente de cuatro fases: la fase de diseño de pruebas, la fase de codificación, la fase de ejecución y la fase de análisis de los resultados.
El objetivo de un proceso de generación de pruebas del sistema es desarrollar las dos primeras fases y obtener esos tres elementos a partir del modelo de requisitos del propio sistema bajo prueba. Dicho proceso toma como punto de partida los requisitos y, a partir de ellos genera los resultados y construye las pruebas. La figura 1 ilustra un proceso genérico que recoge las ideas principales de extraídas después de realizar un estudio comparativo sobre 12 propuestas [6]. Este proceso se describe con mayor detalle en [7]. A partir de este estudio comparativo y de varios casos prácticos, se han identificado un conjunto de actividades pertenecientes al proceso de generación de pruebas de la figura 1 que son independientes de la plataforma de la implementación. Es decir, dichas actividades no se ven afectadas si, por ejemplo, el sistema a prueba es un sistema web o un sistema de escritorio monousuario. De esta manera, es posible generar un conjunto de pruebas independientes de la plataforma. Sólo es necesario conocer los detalles de la plataforma a la hora de implementar las pruebas generadas.

## 4. MODELOS DE PRUEBA

En este punto se describen un conjunto de modelos de prueba independientes y dependientes de la plataforma (PITs y PDTs). Los modelos descritos se muestran en la figura 2.
Los óvalos de la figura 2 representan los distintos modelos implicados. Los óvalos sombreados representan los modelos de requisitos, los óvalos claros representan los modelos independientes de prueba y los óvalos a rayas representan los modelos dependientes. Las líneas entre modelos implican las dependencias y las futuras transformaciones. Todos los modelos siguen el Testing Profile de UML 2.0 [15] siempre que ha sido posible. Los modelos de la figura 2 se describen en los siguientes puntos.

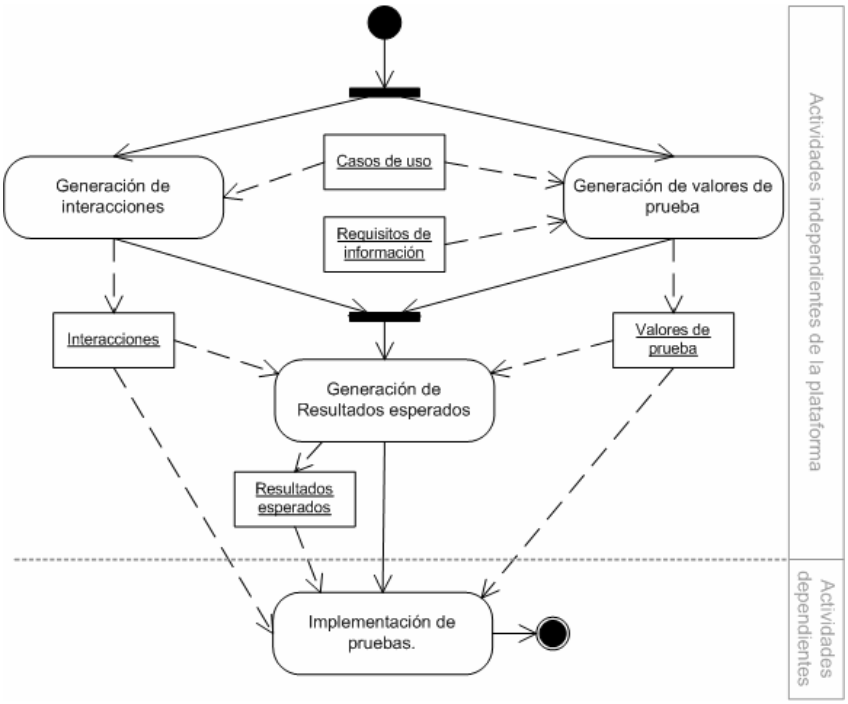


Figura 1. Proceso de generación de pruebas.

A lo largo de este trabajo se muestran ejemplos basados en uno de los casos prácticos realizado durante el desarrollo de estos modelos. El sistema a prueba es una aplicación web para la gestión de un catálogo de enlaces en-línea (www.codecharge.com). En la siguiente sección se definen los modelos de prueba necesarios.

### 4.1. Modelos de requisitos

Los únicos modelos de requisitos necesarios son los casos de uso y los requisitos de almacenamiento, aunque otros modelos, como por ejemplo modelos de interfaces [3] o modelos de navegación [5] pueden enriquecer el proceso de prueba. Actualmente existen varias propuestas de modelos de requisitos. En concreto, la propuesta que utilizamos en este trabajo es Web Requirement (WebRE) [11], la cuál está basada en Navigational Development Techniques (NDT) [5].

### 4.2. Modelo de comportamiento

Un gran número de técnicas de requisitos están basadas en casos de uso definidos en prosa [2]. Uno de ellos es el modelo WebRE utilizado en el punto anterior. Pero no es sencillo manipular programáticamente casos de uso escritos en prosa. Por este motivo, el primer paso de nuestro proceso sistemático de generación de pruebas consiste en expresar dicha prosa mediante un modelo formal manipulable de manera automática.

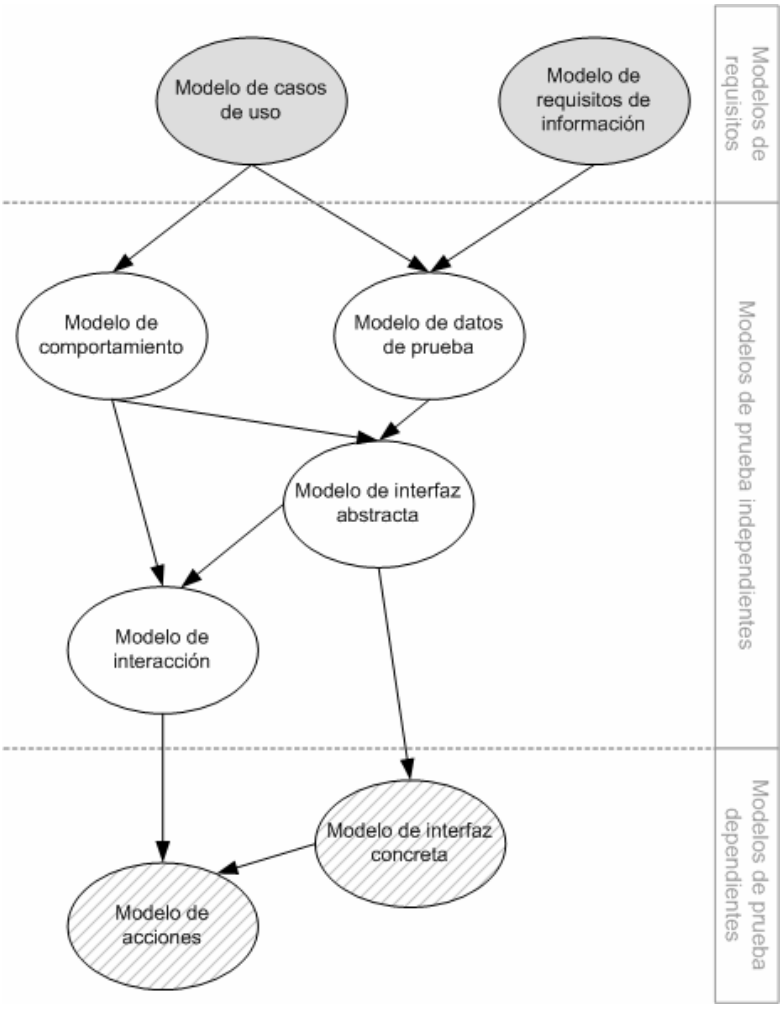


Figura 2. Modelos para la generación de pruebas.

El objetivo del modelo de comportamiento es expresar la misma información contenida en una plantilla de caso de uso de una forma fácilmente manipulable. Las propuestas estudiadas utilizan como modelos de comportamiento diagramas UML de estados, diagramas UML de secuencia o diagramas UML de actividades [6]. En nuestro trabajo hemos seleccionado diagramas de actividades ya que, a diferencia de los diagramas de secuencia, permiten expresar caminos alternativos fácilmente y, a diferencia de los diagramas de estados, permiten expresar la interacción entre el sistema y los actores externos identificando claramente a cada uno de los participantes.

En la tabla 1 se muestra la definición en XML de un caso de uso de una aplicación para gestionar un catálogo de enlaces. En la figura 3 se muestra el diagrama de actividades correspondiente. Dicho diagrama se ha obtenido automáticamente con la herramienta ObjectGen descrita en la sección 4 [8]. Dicha herramienta sólo soporta casos de uso redactados en prosa inglesa. El proceso de generación se describe con más detalle en [8].

```
  <useCase id="01-AddLink">
    <description> This use case allow to introduce a new link. </description>
   <mainSequence>
    <step id="1"> The user selects the option for introduce a new link.
    </step>
    <step id="2"> The System selects top category and shows the form to
introduce the information of a link (SR-02).  </step>
    <step id="3"> The User introduces information of the new link and press
insert button.  </step>
    <step id="4"> The System stores the new link and shows the main screen of
the system.
    </step>
  </mainSequence>
  <alternative>
    <alstep id="3.1.i"> If the user press cancel button then the use case ends.
    </alstep>
    <alstep id="3.2.i"> If the user selects a different category (SR-01) then
system changes the category and the result is to show the form again and execute
step 2.
    </alstep>
    <alstep id="4.1.p"> If the link name or link url is empty, then the system
shows an error message with the result of execute step 2.
    </alstep>
   </alternative>
  </useCase>
```

Tabla 1. Caso de uso en formato XML.

### 4.3. Modelo de datos de prueba

Los casos de uso contienen elementos variables cuyos valores o comportamiento difiere de una ejecución de un caso de uso a otra [1]. Algunos ejemplos son la información suministrada por un actor, una opción seleccionada por un actor, o la información mostrada por el sistema como resultado del caso de uso.

Los objetivos del modelo de datos de prueba son dos. En primer lugar, el modelo de datos de prueba expresa todas las variables del caso de uso [1], su estructura si son tipos complejos (como clientes o compras), las restricciones que puedan existir entre ellos y las particiones de sus respectivos dominios [12]. Esto se realiza mediante un diagrama de clases según la notación propuesta en el Testing Profile de UML [15] (figura 4a). Dicho diagrama de clases puede extraerse automáticamente. Las clases se obtienen a partir de los requisitos funcionales y las distintas particiones se obtienen a partir de las condiciones evaluadas en las alternativas del diagrama de comportamiento. Este diagrama de clases puede refinarse posteriormente añadiendo particiones adicionales si fuera necesario.

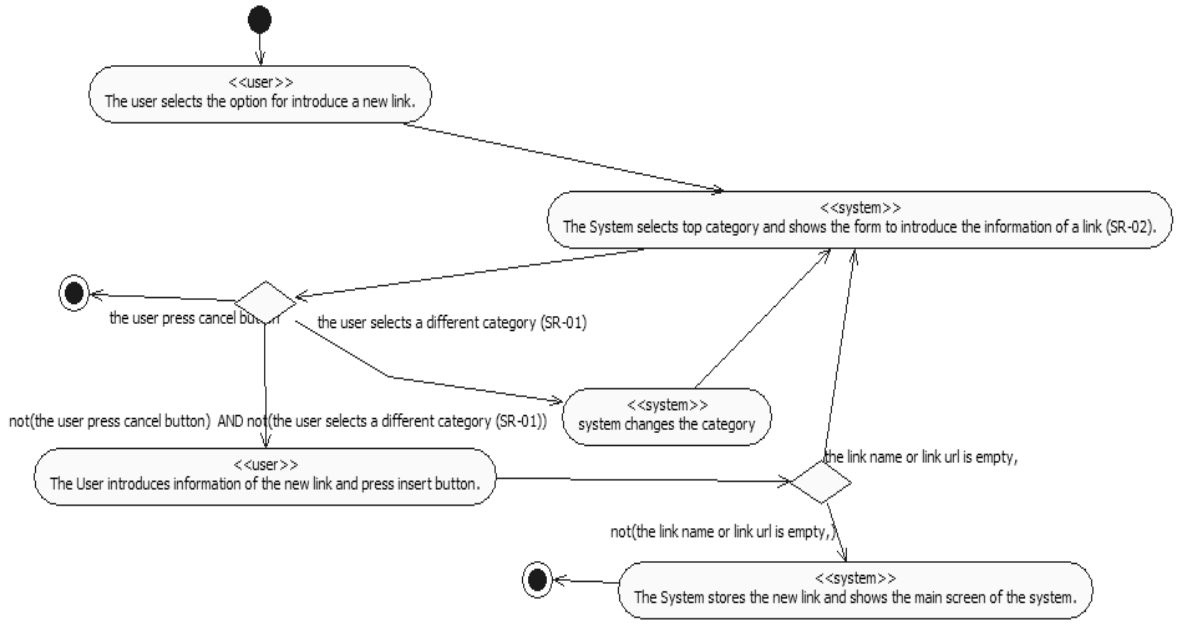


Figura 3. Modelo de comportamiento.

En segundo lugar el modelo de datos de prueba expresa los valores de prueba del sistema y los resultados esperados del mismo. Esto se modela mediante un diagrama de objetos, instanciando las clases identificadas en el modelo de clases anterior (figura 4b).

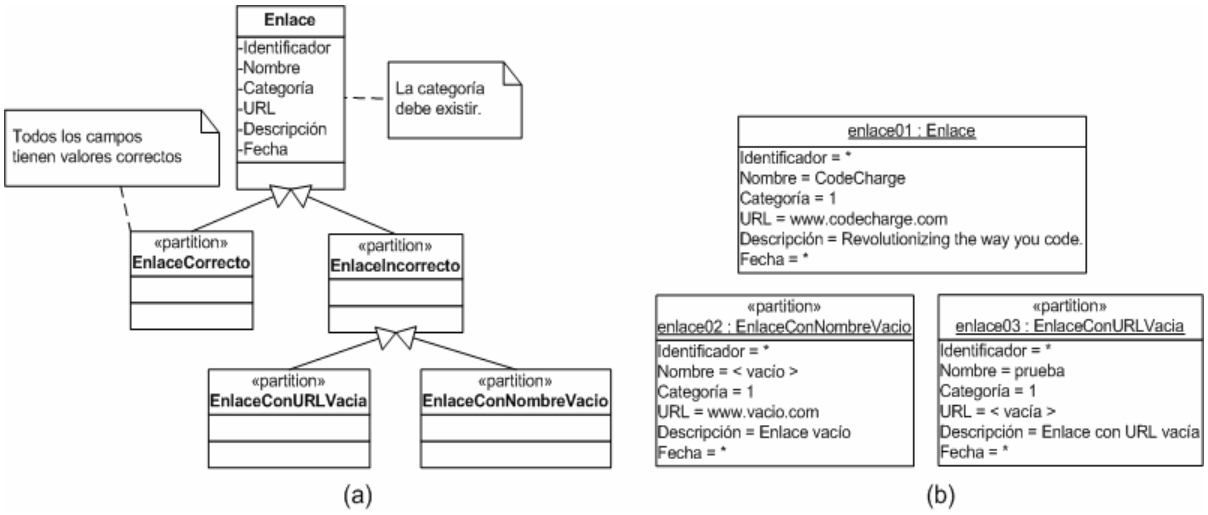


Figura 4. Ejemplo de modelo de datos de prueba.

### 4.4. Modelo de interfaz abstracta

Los modelos anteriores nos indican lo que una prueba debe hacer (ejecutar un escenario posible de un caso de uso), qué información hay que suministrarle y qué información nos va a devolver. Sin embargo estos modelos aún son demasiado abstractos y no se pueden convertir en modelos dependientes de la plataforma ni en pruebas ejecutables de manera directa. Por este motivo, a partir de los modelos anteriores, se obtienen los modelos de interfaz abstracta y de interacción.

Lo primero que es necesario saber para poder implementar las pruebas es cómo interactuar con el sistema, es decir, cuál va a ser la interfaz que el sistema presenta a la prueba para que esta pueda interactuar con él. El objetivo del modelo de interfaz abstracta es definir las interfaces que el sistema ofrecerá para poder realizar la funcionalidad expresada en el modelo de casos de uso y en el modelo de comportamiento. No es necesario, sin embargo, entrar en detalles de la implementación de dichas interfaces.

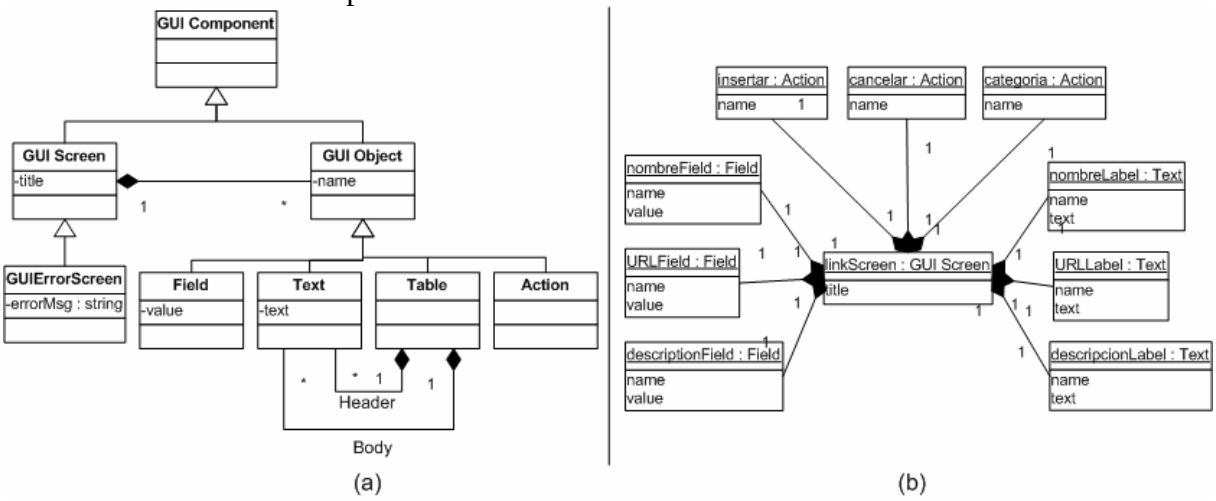


Figura 5. Modelo de componentes para la construcción de interfaces abstractas.

Para lograr la independencia de la plataforma esta modelo se construye a partir de un metamodelo de componentes que abstraen las características de los componentes específicos. Este metamodelo puede adaptarse y ampliarse fácilmente. En la figura 5a se muestra un ejemplo de metamodelo para interfaces gráficas de usuario y, en la figura 5b, un modelo de interfaz abstracta basada en dicho metamodelo.

### 4.5. Modelo de interacción

Una vez que se conocen las interfaces con las que las pruebas interactuarán, expresadas mediante el modelo de interfaz abstracta, se refina el modelo de comportamiento para indicar cómo realizar cada uno de los pasos del caso de uso sobre dicha interfaz.

El objetivo del modelo de interacción es definir cómo realiza las pruebas sus acciones y definir los árbitros. En el contexto de las pruebas, un árbitro es elemento encargado de comprobar si la prueba fue superada o no.

| Instrucción | Descripción |
|---|---|
| ClickOn(component) | Representa una pulsación con el botón izquierdo sobre el componente indicado |
| SetField(field, value) | Asigna al campo el valor indicado. |

Tabla 1. Instrucciones para expresar la interacción del usuario con el sistema.

| Instrucción | Descripción |
|---|---|
| Assert(component.attribute, value) | Verifica que el atributo del componente indicado coincide con el valor. |
| AssertTable(table, index, GUIObject) | Verifica que la fila indicada por *index* de la tabla contiene todos los atributos del objeto en el mismo orden y con el mismo valor. |
| Screen(GUIScreen) | Verifica que la pantalla que muestra el sistema coincide con la pantalla indicada |

Tabla 2. Instrucciones para la definición de árbitros.

Para lograr la independencia tanto de la plataforma como de cualquier herramienta concreta de prueba, se ha definido un conjunto de instrucciones para expresar las acciones entre la prueba y el sistema y las distintas comprobaciones que los árbitros pueden hacer. Dichos conjuntos se muestran en las tablas 2 y 3 y puede ser fácilmente extendido con nuevas instrucciones.

Siguiendo el Testing Profile de UML, los modelos de interacción se expresan mediante diagramas de secuencia. Los árbitros, según su complejidad y reusabilidad, pueden incluirse en dicho diagrama de secuencia o definirse aparte mediante nuevos diagramas de secuencia o diagramas de actividades.

### 4.6. Modelo de interfaz concreta y modelo de acciones

Estos modelos permiten traducir las pruebas abstractas a pruebas ejecutables sobre el sistema. Par ello es necesario conocer las interfaces definitivas, incluir los detalles de dichos interfaces y completar las pruebas abstractas.

El objetivo del modelo de interfaz concreta es expresar los elementos de la interfaz abstracta en función de los componentes concretos del sistema a prueba. A partir de este modelo, ya se pueden expresar las pruebas a nivel de implementación. El objetivo del modelo de acción es expresar los elementos del modelo de interacción mediante un lenguaje de una herramienta de prueba concreta.

Estos modelos dependen de la arquitectura y herramienta de prueba que se utilice para ejecutar las pruebas generadas, dado que las pruebas abstractas deben ser traducidas a pruebas comprensibles por dicha herramienta.

## 4. HERRAMIENTAS

Actualmente se está desarrollando un conjunto de herramientas para la generación de pruebas basada en los modelos descritos en la sección anterior. La primera herramienta

desarrollada se llama ObjectGen [8] y permite convertir un elemento del modelo de casos de uso en un elemento del modelo de comportamiento. Un ejemplo de caso de uso y de diagrama de actividades generado se ha mostrado en la tabla 1 y en la figura 3. Esta herramienta es de código abierto y puede descargarse libremente de www.lsi.us.es/~javierj/

## 5. TRABAJOS RELACIONADOS

Como se ha comentado con anterioridad, existen numerosas propuestas para la generación de pruebas del sistema, sin embargo, no hemos encontrado ninguna propuesta para la generación de pruebas a partir de casos de uso que adopte una aproximación MDA. Algunas de las más relevantes se citan a continuación. Una comparativa más completa puede encontrarse en [6].

En [10] se propone un proceso consistente en desarrollar un diagrama de casos de uso de UML y, a partir de él, construir un diagrama de transición del sistema que refleja las secuencias de casos de uso que pueden ejecutarse. Además, cada caso de uso se modela mediante diagramas de secuencia. Sin embargo esta propuesta no aborda el problema de probar el sistema desde las interfaces gráficas del mismo.

En [14] Se utilizan diagramas de actividades para representar el orden en que deben ejecutarse los casos de uso y, también, se utilizan diagramas de secuencia para representar los pasos de cada caso de uso. Como sus propios autores reconocen, esta es una propuesta incompleta incapaz de generar pruebas ejecutables.

En [1] se identifican las variables de un caso de uso, las cuales son elementos que varían entre dos ejecuciones del mismo, se calculan todos sus posibles valores y se definen los resultados para cada posible combinación. Dado que esta propuesta trabaja sólo con prosa, es difícil de automatizar y no permite la generación de pruebas ejecutables.

Existen también trabajos centrados en pruebas desde una perspectiva en MDA. Por ejemplo, en [9] también se aborda la generación de test oracles, pero no la generación de casos de prueba. Otro trabajo es [13], el cuál define una herramienta llamada MODEST para generar pruebas ejecutables a partir de un modelo de especificación del dominio, en lugar de los casos de uso utilizados en este trabajo.

## 6. CONCLUSIONES

En este trabajo proponemos un conjunto de PITs y PDTs de prueba que permiten desarrollar un proceso de generación de pruebas desde la filosofía de MDA. En este trabajo se han utilizado propuestas existentes como WebRE o UML Testing Profile. Esos modelos son la base para propuestas de generación de pruebas del sistema que deseen automatizar los procesos.

Este trabajo está siendo extendido desarrollando un conjunto de transformaciones para generar los distintos modelos de manera automática. En este sentido, como se ha visto, ya se han establecido el proceso para obtener el modelo de comportamiento a partir de casos de uso y se ha desarrollado una herramienta que implementa dicho proceso.

## REFERENCIAS


[1] Binder, R.V. 1999. Testing Object-Oriented Systems. Addison Wesley.

[2] Cockburn, A. 2000. Writing Effective Use Cases. Addison-Wesley 1st edition. USA.

[3] Dustin E. et-al. 2002. Quality Web Systems. Performance, Security, and Usability. Addison Wesley.

[4] Escalona M.J. Mejías M. Gutiérrez J.J. Torres J. Métodos De Testing Sobre La Ingeniería De Requisitos Web De NDT. Conferencia Iberoamericana IADIS WWW/Internet 2004. 7, 8 Octubre 2004. Madrid.

[5] Escalona M.J. 2004. Models and Techniques for the Specification and Analysis of Navigation in Software Systems. Ph. European Thesis. Department of Computer Language and Systems. University of Seville. Seville, Spain.

[6] 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.

[7] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J. 2006. An Approach to Generate Test Cases from Use Cases. Sixth International Conference on Web Engineering (ICWE'06) Palo Alto, CA. USA. July 11-14

[8] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J. 2006. Derivation of test objectives automatically. Fifteenth International Conference On Information Systems Development (ISD06). Budapest, Hungary, 31 August – 2 September, 2006

[9] Heckel R, Lohmann M. 2003. Towards Model-Driven Testing. Electronic Notes in Theoreical Computer Science 82 no. 6.

[10] 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.

[11] Koch N. Zhang G. Escalona M. J. 2006. Model Transformations from Requirements to Web System Design. Webist 06. Portugal.

[12] Ostrand, TJ, Balcer, MJ. 1988. The Category-Partition Method. Communications of the ACM. 676-686

[13] Rutherford MJ, Wolf AL. 2003. A Case for Test-code Generation in Model Driven Systems. Lecture Notes in Computer Science 2830.

[14] 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.

[15] Object Management Group. 2003. The UML Testing Profile. www.omg.org