Integrando Trazabilidad a la Generación de Casos de Prueba del Sistema: una propuesta MDD/MBT

Natalia Correa[1], Roxana Giandini[1,2]
[1] LIFIA- Laboratorio de Investigación y Formación en Informática Avanzada,
Universidad Nacional de La Plata, 1900 La Plata, Argentina
[2] Universidad Abierta Interamericana, UAI
{natalia.correa, roxana.giandini}@lifia.info.unlp.edu.ar

Abstract. Dentro del contexto de *Model Driven Development*, las transformaciones de modelos permiten generar diferentes modelos con diferentes niveles de abstracción. En tal sentido, cadenas de transformaciones pueden ser codificadas para obtener por, ejemplo, casos de prueba a partir de un modelo de requerimientos. Dada la complejidad de las transformaciones y de los modelos, es necesario contar con técnicas y elementos de modelado que permitan rastrear a aquellos elementos que desde un modelo origen han llevado a la generación de otros en algún modelo destino. El presente trabajo realiza un análisis de trabajos existentes sobre trazabilidad en modelado y presenta una propuesta para integrar trazabilidad a un proceso de generación automática de casos de prueba definido trabajos anteriores. La propuesta se basa en la adaptación para *testing* de un metamodelo para trazabilidad que enriquece al proceso de Transformación de Modelos, permitiendo generar trazas automáticamente durante la ejecución de la transformación.

Keywords: Ingeniería de Software, Trazabilidad, *Model Driven Development*, *Model Based Testing*, Transformaciones de Modelos, Casos de Uso, Casos de Prueba.

1 Introducción

En la Ingeniería de Software, se ha definido trazabilidad como la capacidad de establecer relaciones entre dos o más artefactos de un sistema dentro de un proceso de desarrollo, especialmente aquellos artefactos con vínculos 'predecesor-sucesor' o 'maestro-subordinado' entre sí, por ejemplo, entre los requerimientos y los elementos de diseño que los modelen [1].

La trazabilidad es considerada como una medida de madurez del proceso de desarrollo por varios estándares, algunos de ellos derivados de la metodología 'en cascada' donde era necesario mostrar que el producto final se correspondía con los requerimientos. Luego, CMMI también reconoce esta práctica como ítem de calidad al indicarlo como requisito necesario para alcanzar su nivel 2 [2]. Por lo tanto, podríamos decir que adherir a las técnicas de trazabilidad en los proyectos software es indicativo de una mejor calidad del software producido.

La propuesta MDD –*Model Driven Development*- [3], [4], en cuanto al aporte metodológico y de proceso para la trazabilidad de un sistema, provee la posibilidad de

automatizar la creación y la administración de links -o trazas- mediante la definición y ejecución de transformaciones de modelos. En este contexto, una traza define la relación entre un elemento de un modelo origen y un elemento de un modelo destino. Es así que un artefacto podría rastrearse en la cadena de transformaciones que refinan un sistema mediante el análisis de las trazas generadas.

Adicionalmente, la iniciativa MBT -*Model Based Testing*- [5], [6] dentro del contexto de MDD, es una propuesta reciente que desafía la problemática de generar los modelos y artefactos necesarios para el testeo de software. Su relación con MDD reside en que los modelos de *testing* como los casos de prueba, son derivados total o parcialmente desde modelos origen que describen la funcionalidad del sistema en desarrollo.

Adicionar trazabilidad al proceso de generación de casos de prueba reporta varios beneficios al proceso de desarrollo de sistemas. En la fase de *testing*, brinda la posibilidad de identificar los casos de prueba que son necesarios ejecutar para testear una determinada funcionalidad y permite contar con una medida de la cobertura de los casos de prueba definidos. En el ciclo de vida del software en general, podemos mencionar como ventajas que el modelado de tests -la actividad de testing- puede comenzar tempranamente y que facilita la mantenibilidad de los sistemas.

Nuestro trabajo presenta la generación automática de trazas como parte de la generación de modelos de *testing*, con el objetivo de brindar un aporte a las técnicas de trazabilidad y al *testing* de sistemas. Este trabajo se basa en el establecimiento de trazas entre casos de uso y casos de prueba del sistema dentro de los contextos MDD y MBT. Existen diversas propuestas en este tópico, que serán analizadas en la sección 2. Este análisis nos lleva a entender que una solución deseable requiere tanto automatización y modelización de diferentes tipos de links, así como la posibilidad de definir dinámicamente otros tipos de trazas y contar con el soporte de un metamodelo de elementos para trazabilidad. Es deseable también que puedan utilizarse los beneficios aportados por MDD en cuanto a transformaciones de modelo y las facilidades de los lenguajes de transformación que adhieren nociones de trazabilidad a su sintaxis.

La propuesta actual enriquece trabajos previos de investigación [7], [8] permitiendo completar la definición de un proceso para la generación de casos de prueba del sistema en general y de cada funcionalidad en particular, a partir del modelo de casos de uso, extendiéndolo con trazabilidad.

La organización de este artículo se conforma de la siguiente manera: la sección 2 analiza de distintas propuestas sobre trazabilidad en MDD y en MBT y se los compara con nuestra propuesta. En la sección 3, presentamos una propuesta para la generación de trazas en el contexto MDD/MBT y su integración con trabajos anteriores. La sección 4 introduce un ejemplo, finalizando con la sección 5 la cual expone conclusiones y presenta líneas de trabajo futuro.

2 Trabajos Relacionados

En la sección anterior se han mencionado algunas de las ventajas de contar con técnicas específicas que permitan definir, modelar y manipular trazas entre los

artefactos de un sistema de software. Dada la relevancia que se le ha asignado a la trazabilidad como ítem de calidad de software, estas técnicas han cobrado importancia y también adeptos.

En ese sentido, hemos revisado diferentes informes sobre el estado del arte en MDD en trazabilidad [2], [9] así como procesos, técnicas y propuestas [11] a [22] tanto en esta área como, específicamente, en MBT.

2.1 Propuestas de Trazabilidad en MDD –Model Driven Development–

Existen en la literatura un conjunto de enfoques que definen modelos, metamodelos y frameworks para trazabilidad. A continuación hacemos una breve referencia de algunos de los trabajos más relevantes o representativos de diferentes tipos de aporte.

El lenguaje de modelado UML [10], por medio de la definición de estereotipos –uno de sus mecanismos de extensión-, permite representar relaciones como trazas entre elementos de modelado.

Entre las propuestas que definen metamodelos, podemos mencionar los siguientes trabajos. Amar, Leblanc y Coulette en [11] presentan un metamodelo que permite definir diferentes tipos de links además de contemplar la posibilidad de que los links puedan anidarse. La propuesta de [12] resulta ser amplia y abarcativa que al igual que el trabajo anterior, contempla la creación de diferentes tipos de trazas, pero en este caso, son tipos definidos para todo el ciclo de vida del software desde los requerimientos y hasta el producto final. Otro trabajo en este sentido es [13], donde los autores presentan links tipados y semánticamente ricos. Esto se debe a que las trazas se definen para múltiples metamodelos, cada uno para relacionar tipos determinados de elementos, y que permiten adicionar restricciones en el lenguaje EVL (Epsilon Validation Language).

En cuanto a los lenguajes de transformaciones de modelos propiamente dichos, analizamos el soporte que éstos brindan para generar trazas. Entre ellos, podemos destacar a dos lenguajes de transformaciones: ATLAS y Kermeta. El trabajo presentado por Jouault [14] muestra cómo las trazas pueden ser agregadas a los transformaciones codificados en ATL (ATLAS Transformation Language) [15]. La filosofía del lenguaje considera a las trazas generadas como un modelo así como lo es el modelo de salida resultado de la ejecución de la transformación. Esta idea, permite crear links entre elementos de la misma manera en que son creados otros artefactos. En cuanto al lenguaje Kermeta [16], en [17] los autores toman la propuesta presentada en [14] para ATL y desarrollan un framework para Kermeta con características similares en cuanto a la generación de trazas con el código de la transformación. Agregan además, los conceptos *Trace* y *StaticTrace* que permiten almacenar la traza uno y la traza con los artefactos que relaciona, la otra.

No podemos dejar de mencionar, aunque sea brevemente, al lenguaje estándar CORE de QVT –*Query-View-Transformation*- [18]. Este define la metaclase *Trace*, una clase MOF con propiedades que referencian a artefactos en los modelos que están relacionados por una transformación.

Nuestra propuesta, a diferencia de algunas de estas propuestas que resultan más genéricas por el contexto MDD, propone un metamodelo de trazas con tipos de links

definidos para artefactos de *testing*. Además, utiliza transformaciones de modelo y genera a partir de ellas el modelo de trazas. Este es un modelo aparte de los modelos de origen y destino que forman parte de las transformaciones entre modelos.

2.2 Propuestas de Trazabilidad en MBT –Model-Based Testing–

Distintos trabajos se han estudiado en el contexto MBT, mencionando a continuación aquellas que resultaron más representativas de las soluciones propuestas.

MATERA [19] (y su trabajo relacionado [20]) es un framework que integra el modelado del sistema con la definición de trazas dentro de un proceso MBT. Los requerimientos, modelados en SySML, pueden ser relacionados con otros artefactos que modelan el sistema; con modelos o bien con elementos de los modelos. Cuando los requerimientos se usan para la generación de tests –abstractos, no ejecutables– se asocian como un *tag* al caso de prueba generado. Para tracear los requerimientos a partir de los casos de prueba, se utiliza un script *Phyton* que analiza el *log* del test y genera un *query* en OCL que identifica al requerimiento involucrado en el test.

En [21], la técnica de trazabilidad definida propone un modelo para los requerimientos y la definición de trazas como anotaciones que identifican los elementos que esta relaciona, siendo el objetivo del autor simplificar el proceso de gestión de trazas.

Los requerimientos se especifican en un diagrama de clases semi-formal que incluye dos tipos de elementos: uno es el requerimiento representado como un identificador y el otro, un grupo de clases que representa a los actores del sistema. Estas clases contienen toda la funcionalidad del sistema especificadas como operaciones del diagrama de clases. Las trazas se definen como anotaciones con *tags* de identificación, técnica elegida de entre las muchas que la trazabilidad en requerimientos propone. La forma en que se gestiona la trazabilidad es manteniendo una relación (que contiene el *tag* identificatorio del requisito) entre el modelo de requerimientos mencionado y el diagrama de estados que se modela como representación del comportamiento del sistema.

Por último, un trabajo más actual en el tiempo [22], propone el uso de tecnologías XML para la definición de trazabilidad en MBT. En este caso, la traza es una relación elemento a elemento definida con una estructura XML formal: RDML (*Relation Definition Markup Language*). Para cada tipo de modelo utilizado, una definición RDML -basado en un esquema XML- especifica la relación con otro modelo y su tipo, siendo el tipo “derivado” o bien, “referenciado”.

Si bien esta idea permite una rápida definición de la traza y cuenta con la facilidad de estar basada en un esquema XML, carece de especificación del tipo de traza.

Nuestra propuesta se diferencia de estos trabajos en varios aspectos. Para el modelado de requerimientos, elegimos casos de uso que tienen la ventaja de pertenecer a un estándar (UML), tener buen soporte de herramientas y amplia difusión entre los analistas.

Por otra parte, nuestro enfoque propone un metamodelo de trazas no genérico, sino específico para artefactos de *testing*. Como ventaja, podemos decir que los tipos de trazas que definimos son específicos del contexto que interesa dentro del proceso MBT. Además, contar con un modelo de trazas que conforma un metamodelo

favorece la realización de los análisis y validaciones sintácticas automáticas que pueden realizar distintas herramientas.

Otra diferencia es que nuestra propuesta genera a partir de las transformaciones de modelo, el modelo de trazas por separado de los modelos de origen y destino, otorgándole a la traza identidad como elemento de modelado y evitando de esta forma la 'polución' de los modelos origen y/o destino.

Con esta revisión sobre el estado del arte de Trazabilidad en los contextos que nos interesan, MDD/MBT y los lenguajes de transformaciones de modelos, podemos introducir nuestra propuesta.

3 Una propuesta para Generación Automática de Trazas

Inicialmente, nuestro trabajo [7], [8] presenta un proceso para generar casos de prueba a partir de los casos de uso, teniendo en cuenta que por definición, un caso de uso retorna un valor al usuario y que por lo tanto, éste puede ser verificado. En una visión más general del proceso, puede decirse que la actividad de *testing* puede ser organizada y guiada por los casos de uso ya que una vez que estos han sido definidos, es posible generar los casos de prueba correspondientes por medio de las transformaciones de modelos.

En esta sección, proponemos integrar trazabilidad al proceso de *testing* desarrollado en los trabajos anteriores basándonos en las propuestas analizadas en la sección 2.

3.1 Metamodelo para Trazas de *Testing*

Del análisis realizado y especificado en la sección 2, concluimos que nuestra propuesta, al modelar trazas, debe contar con un metamodelo que lo soporte. Este metamodelo debe contener los tipos de links predefinidos que son necesarios especificar para notar las trazas, además de brindar la posibilidad de definir nuevos tipos. De esta forma, se unifica criteriosamente a los tipos de trazas identificados y se permite la configuración de otros diferentes a los establecidos.

En tal sentido, el trabajo presentado por Amar, Leblanc, y Coulette [11] se acerca más a los objetivos de nuestra propuesta, el cual se modificamos para modelar trazas que relacionen artefactos de *testing*.

La figura 1 presenta el metamodelo para trazabilidad mencionado con la extensión para *testing* en recuadro punteado sobre el lado izquierdo.

El metamodelo define que una transformación se compone de un conjunto de trazas las cuales relacionan elementos del modelo origen y elementos del modelo destino. La noción de traza puede ser simple o compuesta, lo cual permite representar cadenas de transformaciones o trazas anidadas. Este hecho se verifica al definir transformaciones que llaman a otras. En nuestro caso, un caso de prueba puede requerir casos de pruebas de datos de los parámetros ingresados.

La adición de tipos de links tiene el objetivo de especificar los tipos de *tests* que pueden relacionar a los requerimientos con los artefactos de *tests*. En nuestro caso, los casos de prueba del sistema presentan *tests* de integridad entre las diferentes

funcionalidades del sistema, cada caso de uso tiene sus correspondientes casos de prueba y los datos de ingreso, cuentan con sus casos de prueba de datos. Para cualquier especificación adicional de traza que requiera ser modelada, se cuenta con el elemento genérico *AnyLinkType*.

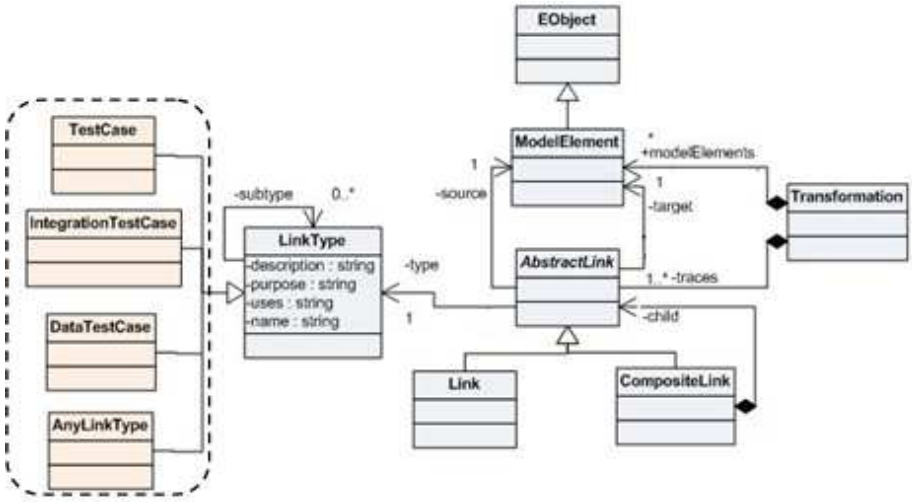


Fig. 1. Metamodelo para trazabilidad con artefactos de *testing* (recuadro punteado)

3.2 Generación Automática de Trazas

Adicionalmente y en cuanto a técnicas de generación de trazas, podemos citar dos grandes grupos según lo menciona Aizenbud-Reshef en [9]: los de trazas inferidas y los de trazas impuestas. El primer grupo hace referencia a las metodologías que crean trazas a partir de artefactos de software existentes y que cumplen con ciertas reglas o condiciones para poder ser relacionados. Las trazas impuestas, por su parte, son las creadas voluntariamente entre artefactos nombrados explícitamente. Nuestra propuesta pertenece al grupo de trazas impuestas ya que genera el modelo de trazas a partir de la ejecución de una transformación entre modelos origen y destino en lugar utilizar técnicas de inferencia que, según el caso, deberían analizar los artefactos o el código para encontrar situaciones como que una operación llame a otra y por ello genere una relación de dependencia –traza- entre estos elementos.

A continuación presentamos parte de la transformación que genera los elementos del modelo destino junto con las trazas. La transformación y sus reglas fueron presentadas en trabajos anteriores [7], [8] destacándose en el presente, sólo la adición de la generación de trazas. La misma se presenta en pseudocódigo “like ATL [15]” por razones de espacio y para aportar mayor legibilidad al ejemplo presentado.

```
Transformation uc2Activity (UML uml, TraceModel traceM)
Input uml:uc
Output uml:activity, traceM:trace
```

```
{activity ←UML::Activity.new
 activity.name ← “ad” + uc.name
 initialNode:= UML::InitialNode.new
 activity.node.add (initialNode)
 …
 for each uc.actor in uc{
       partition ←UML.Partition.new
       partition.name ← uc.actor.name
       partitions.add (partition)
       invokeRule createTraces (uc.actor, partition,
                          AnyLinkType.new(“actor2Partition”));
 }
 …
for each uc.useCase in uc{
       activity ←UML.Activity.new
       activity.name ← uc.name
       activity.node.add (activity)
       invokeRule createTraces (uc.useCase, activity,
       IntegrityTestCase.new ());
  }
 …
for each uc.relationship in uc{
       if (uc.relationship.isTypeOf (“include”))
              edge ← UML::ActivityEdge.new
              edge.stereotype ← “include”
              activity.edge.add (edge)
       else if (uc.relationship.isTypeOf (“extend”))
       …}
…
finalNode:= UML::FinalNode.new
activity.node.add (finalNode)
}

Transformation Rule createTraces (trace_source, trace_target,
trace_type) {
  traceLink ←CompositeLink.new
  traceLink.source ← trace_source
  traceLink.target ← trace_target
  traceLink.type ← trace_type
  …
}
```

La figura 2 muestra gráficamente parte del trabajo anterior con la actual integración de trazas. Aquí se muestra el modelo de casos de uso a partir del cual se generan el modelo intermedio DATS -Diagrama de Actividades de *Testing* del Sistema- y el modelo de trazas, obtenidos ambos de la primera transformación definida en nuestro proceso. Luego y a partir del DATS se generan mediante otra transformación los casos de prueba del sistema.

Cabe mencionar que el pseudocódigo y la figura 2 se encuentran relacionados. El código explicita la transformación que genera dos de los modelos que la figura 2

muestra: el DATS y el modelo de trazas. El modelo de casos de uso es el modelo origen de nuestro proceso y de nuestra transformación.

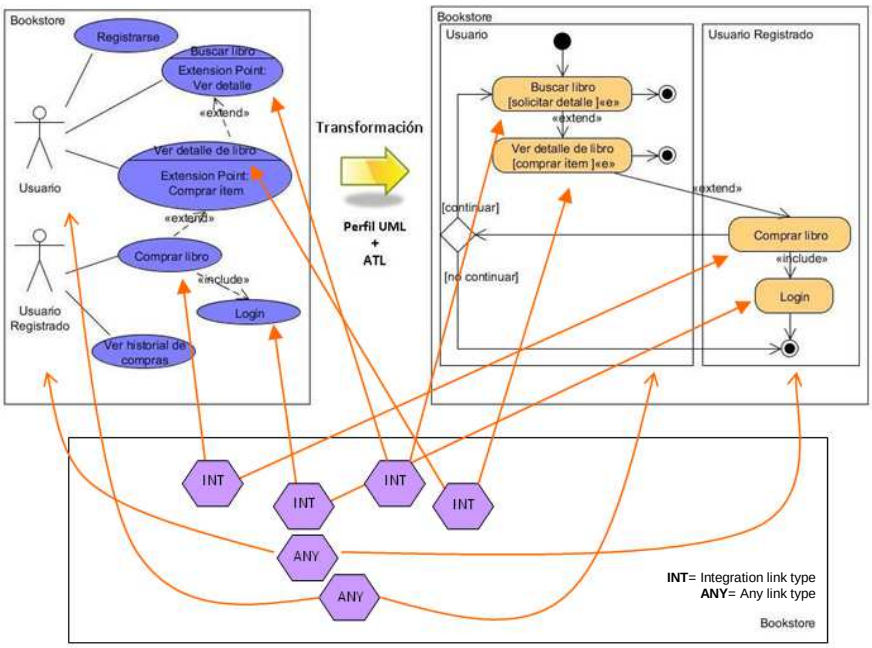


Fig. 2. Modelos de Casos de Uso, DATS y trazas generados a partir de la transformación

3.3 Integración de esta Propuesta con las Anteriores- Generación de Casos de Prueba a partir de Casos de Uso-

A continuación y sirviéndonos de otra imagen como la presentada en la figura 3, mencionamos cómo se da la integración de la trazabilidad al proceso de generación de casos de prueba.

A partir del modelo de casos de uso, las propuestas anteriores generan:

- un DATS a nivel de sistema: por medio de una transformación entre modelos, donde se explicitan las ejecuciones que los usuarios pueden realizar sobre el sistema. A partir del DATS se generan los casos de prueba del sistema, o sea, todos los caminos de ejecuciones, a través de una transformación M2T –modelo a texto-. Estos casos de prueba conforman un esquema general sin estar asociados a ningún lenguaje en particular; o sea, no son aún *tests* ejecutables
- un diagrama de actividades de test: a partir de una transformación desde cada caso de uso y su especificación. Luego, desde ese modelo intermedio, se generan por medio de otra transformación, los casos de prueba finales.

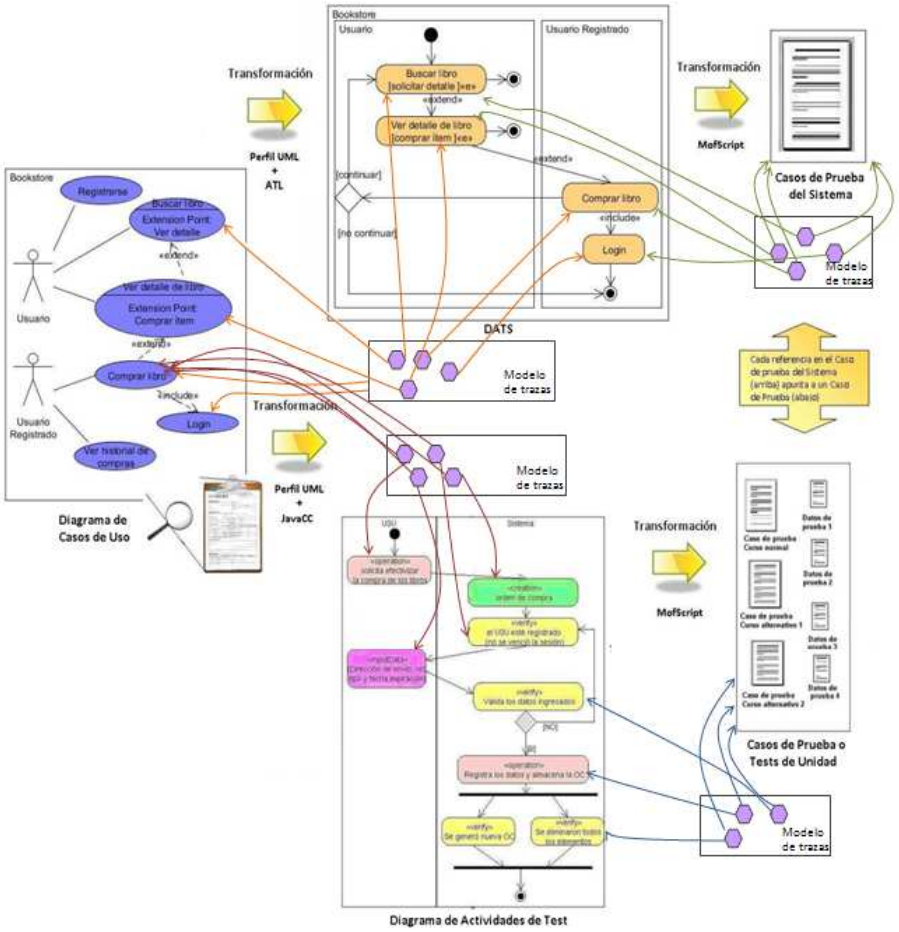


Fig. 3. Proceso completo de generación de casos de prueba a partir de casos de uso

A este proceso ya definido se le agregan las nociones de trazabilidad mencionadas en el presente trabajo. La generación de los modelos de trazas, obtenidos con cada una de las transformaciones mencionadas, permiten establecer links entre los distintos artefactos de los modelos involucrados, por ejemplo, entre un caso de uso, la actividad de test y el caso de prueba que lo contiene.

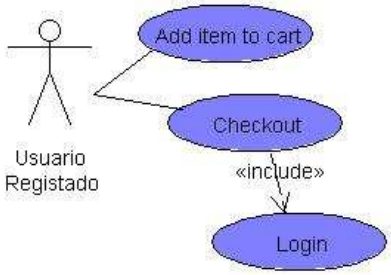
4 Ejemplo

El siguiente ejemplo ilustra el uso del proceso definido para la generación de casos de prueba con la integración de la trazas. El ejemplo se basa en un sistema de venta de libros por Internet, al estilo Amazon, que permite buscar, ver detalles de los libros y por supuesto, comprarlos. El enunciado completo, que por razones de espacio no

detallamos completamente, fue tomado de [4]. En las siguientes subsecciones se muestran los pasos del proceso definido previamente, aplicado a este ejemplo.

4.1 Definición de los Casos de Uso del Ejemplo

Se definen sólo tres (3) casos de uso para no agregar complejidad innecesaria a esta parte del ejemplo. Acompaña a la figura, la documentación extendida al "estilo Larman" [23] del caso de uso "Checkout", como puede verse en la Figura 4. La definición completa de los casos de uso y sus descripciones puede consultarse en el informe técnico [24].



Nombre: Checkout
Actor: usuario (USU)
Descripción: El usuario solicita efectivizar la compra de los libros agregados a su carrito de compras.
Precondición: El carrito tiene 1 ó más ítems. El usuario está registrado en el sistema.
Postcondición: Se generó una nueva orden de compra. Se eliminaron todos los elementos del carrito de compras.

Curso Normal
1- El USU solicita efectivizar la compra de los libros de su carrito
2- El sistema recupera el carrito de compras de la sesión de USU
3- El sistema crea una orden de compra
4- El sistema valida que el USU esté registrado (no se venció la sesión)
5- El sistema agrega los ítems del carrito a la orden de compra (OC)
6- El sistema muestra la OC y el precio total a pagar
7- El sistema solicita al cliente dirección de envío y los datos de la tarjeta de crédito
8- El usuario ingresa la dirección de envío, los datos de su tarjeta de crédito y confirma la compra
9- El sistema valida los datos ingresados
10- El sistema registra los datos y almacena la OC

Curso Alternativo- Usuario no logueado
3.1-El usuario no está logueado o se venció su sesión de usuario -> ejecutar CU Login

Curso Alternativo- Datos inválidos
9.1-Los datos ingresados no son válidos -> informar y volver al paso 7

Fig. 4. Algunos casos de uso del ejemplo Bookstore y documentación de Checkout

4.2 Transformación de los Casos de Uso en DATS y en Diagramas de Actividades de *Testing*

A partir del modelo de Casos de Uso definido, se trabaja en dos líneas:
- con cada caso de uso
- con todos los casos de uso definidos en el modelo

En el primer caso y por medio de una transformación, se genera un diagrama de actividades intermedio enriquecido con un perfil para *testing* de forma tal de obtener un diagrama con la funcionalidad descripta en el caso de uso y los elementos

necesarios para verificarla. Este paso es parte del trabajo presentado en [7]. En este diagrama los analistas pueden intervenir y agregar actividades para completar aquello que sea necesario, en caso de que se haya omitido por alguna razón, en la documentación del caso de uso.

En el segundo caso, los casos de uso presentados (DCU) conforman el modelo de entrada de la transformación que permite generar el *DATS* del sistema Este paso es parte del trabajo presentado en [8].

En cada una de las transformaciones definidas, se agregan las reglas necesarias para la obtención de las trazas entre los distintos artefactos.

Las transformaciones mencionadas pueden consultarse en el informe técnico [24].

4.3 Transformación del Diagrama de Actividades de Testing a Caso de Prueba y del DATS a Casos de Prueba del Sistema

Una vez obtenidos los diagramas de actividades y DATS, modelos intermedios para la obtención de casos de prueba, se realizan las siguientes actividades: con el diagrama de actividades de *testing* y a través de una transformación modelo a texto, se generan los casos de prueba: uno por el curso normal y uno por cada curso alternativo. La transformación define, para cada tipo de actividad, una sentencia determinada según lo especificado en el trabajo [7]. A partir del presente trabajo se le adiciona la generación de trazas.

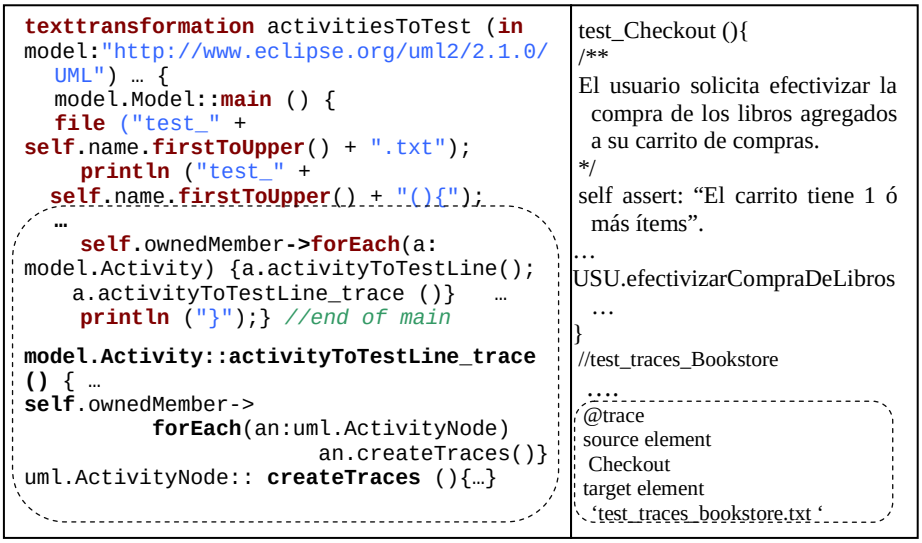


Fig. 5. Transformación en MOFscript y el caso de prueba como modelo de salida

La figura 5 se encuentra dividida en dos partes y muestra: en la columna izquierda, parte de la transformación definida en MOFscript, y en la columna derecha, parte de uno de los casos de prueba generados a partir del curso normal detallado en la documentación del caso de uso.

Por otra parte, en el diagrama de actividades (DATS) obtenido, se mantienen las relaciones semánticas explicitadas en el modelo de casos de uso. Nuestro DATS modela los posibles caminos que un actor puede ejecutar en el sistema. Testear esos caminos de ejecución brinda la ventaja de conocer si una secuencia de acciones puede ser realizada o no en el sistema; o bien, si ese camino es exitoso, falla, o genera un error.

Luego de la ejecución de la transformación, se obtiene un archivo con todos los caminos posibles que un usuario puede realizar sobre el sistema, contemplando los posibles ciclos de ejecución. La figura 6 muestra parte del código de la transformación en MOFscript y parte del listado de los posibles caminos de ejecución para el modelo de casos de uso del ejemplo.

La definición de cada una de las transformaciones, cuenta con la adición de la generación de links entre los artefactos origen y destino. En este caso, las actividades de *testing* se relacionan con los casos de prueba generados.

Fig. 6. Código de la transformación y casos de prueba del sistema generados

El código de la transformación completo, que por razones de espacio no se muestra en su totalidad, puede ser consultado en [24].

5 Conclusiones y trabajo futuro

En este trabajo hemos presentado una propuesta para representar trazabilidad en el contexto MDD/MBT.

Se definió una extensión con conceptos de *testing* a un metamodelo para trazabilidad existente. En el mismo, se definen los tipos de links necesarios para modelar las trazas permitiendo además definir otros tipos según la necesidad del usuario. Estos tipos de trazas son específicos al contexto de *testing* en MBT. Asimismo, se extendieron las transformaciones de modelos que permiten generar modelos de tests (definidas en trabajos anteriores) con la generación de las trazas.

Como ventajas, podemos indicar que nuestra propuesta permite que se cuente con un modelo separado de los modelos de origen y destino para las trazas, evitando sobrecargar los modelos con información referente a la trazabilidad. También se favorece la realización de los análisis y validaciones sintácticas automáticas por parte de diferentes herramientas al contar con un modelo –el de trazas- conforme a un metamodelo definido y le aporta asimismo, automaticidad al proceso. En una visión más amplia, podemos decir que nuestro trabajo realiza un aporte a los procesos de software guiados por casos de uso mejorando el soporte al inicio temprano de la fase de *testing*. Además, la incorporación de trazas a nuestro proceso adiciona las ventajas propias del mecanismo de trazabilidad: implícitamente se cuenta con una métrica de análisis de impacto al poder determinar cómo los cambios en un artefacto inciden en cambios en otros con los cuales se relacionan; permite conocer la cobertura de requerimientos que cuentan con sus casos de prueba definidos; y favorece la sincronización y la consistencia entre los modelos generados y sus artefactos.

Como desventaja, podemos mencionar que ante la aparición de un cambio, siempre se ejecuta la transformación completa para generar los modelos modificados, sin posibilidad de que pueda ejecutarse, y por ende actualizar, solo la parte que interesa por el cambio introducido. En modelos extensos y en transformaciones complejas, esto acarrea cierta cantidad de procesamiento y actualización innecesaria.

En cuanto a las líneas de trabajo futuro, se espera definir una herramienta de soporte que permita automatizar completamente el proceso –el trabajo actual y los anteriores-, permitiendo aplicar al Diagrama de Casos de Uso (DCU) las transformaciones que generen los diagramas de actividades intermedios con conceptos de *testing* y sus consecuentes –segundas- transformaciones a casos de prueba y casos de pruebas del sistema, con la adición de las trazas entre los requerimientos, los modelos de test intermedios y los casos de prueba generados.

El editor gráfico será creado como *plugin* para Eclipse [25] con EMF [26] y GEF [27]. JavaCC [28] y el lenguaje ATL se utilizarán como herramientas auxiliares para las transformaciones desde un caso de uso al diagrama de actividades intermedio en un caso y, en el otro caso, para la transformación desde un DCU al DATS.

Referencias

1. Geraci, A.: IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries. Institute of Electrical and Electronics Engineers Inc. (1991)

2. Galvão, I., Goknil, A.: ”Survey of Traceability Approaches in Model-Driven Engineering”. IEEE International EDOC Enterprise Computing Conference (2007).
3. Kleppe, A., Warmer J., Bast, W.: “MDA Explained: The Model Driven Architecture:”
4. Pons, C., Giandini, R., Pérez, G.: “Desarrollo de Software Dirigido por Modelos. Conceptos teóricos y su aplicación práctica”. EDULP & McGraw-Hill Educación. (2010).
5. Baker, P. et all: “Model-Driven Testing Using the UML Testing Profile”. Springer-Verlag
6. Blackburn, M., Busser, R., Nauman, A.: ”Why model-based test automation is different and what you should know to get started”. International Conference of Practical Software Quality & Testing (2004).
7. Correa, N., Giandini, R.: “Generación Automática de Casos de Prueba a partir de Casos de Uso: Una Propuesta Basada en MDD/MDT”. ASSE 2011. Argentine Symp on 40 JAIIO (2011)
8. Correa, N., Giandini, R.: “Casos de Prueba del Sistema Generados en el Contexto MDD/MDT”. ASSE 2012. Argentine Symp on 41 JAIIO (2012)
9. Aizenbud-Reshef, N et all.: , A.: ”Model Traceability”. IBM Systems Journal, vol.45 nro. 3, pág. 515–526 (2006).
10. UML 2.3. The Unified Modeling Language Superstructure version 2.3. OMG Final Adopted Specification. http://www.omg.org (2010).
11. Amar, B., Leblanc, H., Coulette, B.: ”A Traceability Engine Dedicated to Model Transformation for Software Engineering”. Traceability Workshop of the European Conference on MDA (2008).
12. Letelier, P.: ”A framework for requirements traceability in UML-based projects”. Intl. Workshop on Traceability in Emerging Forms of Software Engineering (2012).
13. Drivalos, N., Paige, R., Fernandes, K., Kolovos, D.: "Towards rigorously defined model-to-model traceability". Traceability Workshop of the European Conference on MDA (2008).
14. Jouault, F.: ”Loosely Coupled Traceability for ATL”. Traceability Workshop of the European Conference on MDA (2005).
15. ATL (ATLAS Transformation Language) http://www.eclipse.org/m2m/atl/
16. Triskel project (IRISA). The Metamodeling Language Kermeta. http://www.kermeta.org
17. Kolovos, D., Paige, R., Polack, F.: ”Merging Models with the Epsilon Merging Language”. Conference on Model Driven Engineering Languages and Systems (Models/UML’06) (2006).
18. MOF QVT final adopted specification. Technical Report ptc/05-11-01, OMG (2005).
19. Abbors, F., Bäcklund, A., Truscan, D.: “MATERA - An Integrated Framework for Model Based Testing”. IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (2010)
20. Abbors, F., Bäcklund, A., Truscan, D.: “Tracing Requeriments in a Model-Based Testing Approach”. First International Conference on Advances in System Testing and Validation Lifecycle (2009)
21. Pasupulati, B. “Traceability in Model Based Testing”. Tesis de Master of Software Engineering. School of Innovation, Design and Engineering. Sweden (2009) http://www.idt.mdh.se/utbildning/exjobb/files/TR0962.pdf
22. George, M. et all: “Traceability in Model-Based Testing”. Article of Future Internet.An Open Access Journal from MDPI (2012). www.mdpi.com/journal/futureinternet
23. Larman, C.: “Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development”. Prentice Hall, 3era. ed. (2004).
24. Informe técnico. Generación de casos de Prueba https://sol.lifia.info.unlp.edu.ar/~nataliac/ (2012)
25. The Eclipse Project. http://www.eclipse.org/.
26. Eclipse Modeling Framework EMF. http://www.eclipse.org/modeling/emf/
27. Graphical Editing Framework GEF. http://www.eclipse.org/gef/
28. Java Compiler Compiler. http://javacc.java.net/