Análisis de consistencia de Casos de Uso con simuladores de autómatas finitos.

Marcelo Martín Marciszack, Oscar Carlos Medina, Claudia Castro, Enrique Humberto Moyano

GIDTSI, Grupo de Investigación, Desarrollo y Transferencia de Sistemas de Información
Universidad Tecnológica Nacional, Facultad Regional Córdoba
Maestro López esq. Av. Cruz Roja Argentina, Ciudad Universitaria – (5016) Córdoba

{marciszack, oscarcmedina, ingclaudiacastro, enriquemoyano}@gmail.com

Abstract
El objetivo del trabajo es obtener un método automatizado de análisis de consistencia de Casos de Uso. Para lo cual se construyó una aplicación web denominada SIAR (Sistema Integral de Administración de Requerimientos) que gestiona los requerimientos funcionales de un sistema de información según los lineamientos de UML (Lenguaje Unificado de Modelado).
Los casos de uso son una herramienta de generación y análisis de requisitos de sistemas. La finalidad principal de SIAR es la administración de casos de uso con una herramienta informática que agilice su registración, normalice su contenido y posibilite implementar validaciones funcionales, como por ejemplo un método automatizado de análisis de consistencia de casos de uso. Con este fin, el sistema genera un grafo con la transición de estados de cada caso de uso, expresado en el protocolo XPDL (Lenguaje de Definición de Flujo de Trabajo), que es analizado en un simulador de autómata finito determinista para verificar la cohesión de los escenarios en él definidos.

Palabras Clave
Caso de uso, Análisis de requerimientos, Autómata finito determinista, UML, XPDL.

Introducción

En la actualidad UML es reconocido como el estándar para el modelado de proyectos de software orientado a objetos.
Los componentes principales de esta metodología son los Casos de Uso que especifican el comportamiento deseado del sistema. Por definición el caso de uso "especifica una secuencia de acciones, incluyendo variantes, que el sistema puede ejecutar y que produce un resultado observable de valor para un particular actor" [1].
Una vez generado un caso de uso, es necesario comprobar y asegurar su validez. La verificación de consistencia de la secuencia de acciones descripta en el caso de uso es una de las tareas que permiten su validación.
Es deseable que esta verificación pueda realizarse de manera automatizada para lo cual se podría trabajar con autómatas finitos deterministas, ya que un autómata finito es un conjunto de estados y un control que se mueve de un estado a otro en respuesta a entradas externas [2]. Se llama determinista al autómata que puede estar únicamente en un estado en un momento determinado. El desafío es transformar el caso de uso en un autómata finito determinista para validar su cohesión secuencial.
Partiendo de estas premisas, el aporte que pretende hacer el presente análisis consiste en dar respuesta a la siguiente pregunta:
¿Cómo se definiría un método, basado en simuladores de autómatas finitos deterministas, que verifique la consistencia de la secuencia de acciones definida en un caso de uso, incluyendo variantes?.

Elementos del Trabajo y metodología

A continuación se presenta la metodología con una propuesta de tres fases:

a) Normalizar la registración del requerimiento
b) Transformar el caso de uso en una máquina de estados

c) Verificar la consistencia secuencial de los escenarios del caso de uso

a) Normalizar la registración del requerimiento:
Un requerimiento es una característica de diseño, una propiedad o un comportamiento de un sistema. Cuando se enuncian los requerimientos de un sistema se está estableciendo un contrato entre los elementos externos al sistema y el propio sistema, que establece lo que se espera que haga el sistema. Los requerimientos se pueden expresar de varias formas, desde texto sin estructura hasta expresiones en un lenguaje formal, pasando por cualquier otra forma intermedia. La mayoría de los requisitos funcionales de un sistema, si no todos, se pueden expresar con casos de usos [1].
Como el modelo de requerimientos tiene por objetivo la captura de requerimientos funcionales [3], surgió la necesidad de una herramienta propia que se llamó SIAR (Sistema Integral de Administración de Requerimientos), con el fin de brindar soporte a la construcción del modelo de requerimientos de un proyecto de software según el estándar UML. El desarrollo se programó con la plataforma GeneXus versión X evolution II por su alta eficiencia y flexibilidad para gestionar cambios, y la disponibilidad de licencias para uso académico de acuerdo al programa universitario de extensión celebrado con la empresa Artech.
SIAR es una aplicación web que permite registrar en forma normalizada los casos de uso y cuya primera versión comprende el siguiente alcance:

- Administración de los atributos de un proyecto de sistemas y sus versiones.

- Gestión de los alcances de cada versión del proyecto y los casos de uso asignados.

- Administración de los artefactos de un caso de uso, incluyendo actores, pre-condiciones, post-condiciones, escenario principal y escenarios alternativos, y su versionado.

- Clasificación, priorización y trazabilidad de los casos de uso.

- Visualización de consultas y generación de reportes en distintos formatos, inclusive XPDL, para comunicarse con otras aplicaciones.

- Diseño y validación del Modelo Conceptual.

- Gestión de atributos de procesos de negocio, de actividades de negocio que los componen y los casos de uso asociados a estas actividades.

- Registración y consulta de un glosario por proyecto, con entradas y sinónimos, siguiendo las recomendaciones de LEL (Léxico Extendido del Lenguaje), que es una estrategia de modelado de requisitos basada en lenguaje natural [4], que tiene por objetivo "comprender el lenguaje del dominio del problema y luego estudiar la dinámica de éste" [10].

Figura 1: Versionado de Proyectos y Casos de Uso en SIAR

b) Transformar el caso de uso en una máquina de estados:
En segunda instancia se definió un conjunto de reglas de conversión del caso de uso en un grafo de estados para que sea la entrada a un simulador de autómata finito determinista.
Una vez completa la versión de un caso de uso, se genera un grafo de estados a partir de las siguientes definiciones:

- Todo caso de uso tiene al menos un escenario principal.

- Un caso de uso puede no tener ningún, o tener uno, o tener varios escenarios alternativos.

- Cada paso de un escenario de un caso de uso responde a un estado y es un nodo de la máquina de estados.

- Todo caso de uso tiene un paso inicial, y es el primer paso de todos los escenarios. Este paso es el estado/nodo inicial.

- Todo caso de uso tiene un paso final por aceptación y es el último paso del escenario principal. También puede ser el último paso de algún escenario alternativo. Este paso es el estado/nodo final por aceptación.

- Un caso de uso puede no tener ningún final por error, o tener uno o varios finales por error, en cuyo caso son el último paso de un escenario alternativo. Estos pasos se denominan estados/nodos finales por error.

- Todo caso de uso pasa de un estado/nodo a otro por medio de una función de transición, que es una tabla de transición de estados donde se muestra a qué estado se moverá un autómata finito dado, basándose en el estado actual y otras entradas.

- Partiendo de un estado/nodo origen se puede continuar en un único estado/nodo destino, o en dos nodos/estados destino alternativos dependiendo de una condición de bifurcación.

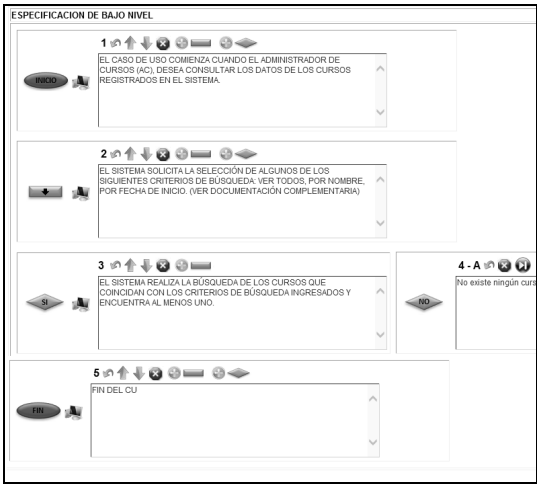

Figura 2: Bifurcación del escenario principal en la especificación de bajo nivel de un caso de uso.

- El grafo de estados asociado al caso de uso tiene un alfabeto de tres símbolos para indicar qué evento lo cambia de un estado/nodo a otro:
  - Símbolo "A" = Por medio de una Acción determinada.
  - Símbolo "S" = Cuando Si se cumple una condición que bifurca a un escenario alternativo.
  - Símbolo "N" = Cuando No se cumple una condición que bifurca a un escenario alternativo.

- Partiendo de un estado/nodo origen, en la función de transición puede estar asociado solamente uno de los símbolos: "A", "N" o "S". Con esto se cumple la condición necesaria de un autómata finito determinista.
  De esta manera, si la transición entre dos estados/nodos se da dentro de un mismo camino, se asocia el símbolo "A". Si en cambio interviene una bifurcación, la función de transición hacia el estado/nodo destino por cumplimiento de la condición de

bifurcación, se asocia el símbolo “S”. Por el otro camino de la bifurcación, se asocia el símbolo “N”.

Tabla De Estados Casos de Uso: 1 - 1 - 1 - 1 CONSULTAR CURSOS Versión: 6

Estado / Paso Origen	Estado / Paso Destino	Transición	Estado Final por Error	Tipo
1	2	A		S
2	3	A		S
3	4	A		S
4	4 - A	N		C
4 - A	4 - A - 1	A		S
4 - A - 1	4 - A - 2	A	SI	S
4	5	S		C
5	6	A		S
6	6 - A	S		C
6 - A	6 - A - 1	A		S
6	7	N		C

Figura 3: Tabla de estados de un caso de uso. Transformación automática por SIAR.

- Una vez generado el grafo de estados se expresa en protocolo XPDL, por ser el más adecuado para intercambiar modelos de procesos entre distintas herramientas. Este protocolo de intercambio de datos está basado en XML y su sintaxis fue creada específicamente para representar el diagrama de flujo de un proceso.

```
<transicion
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="java:dominio.automatas.TransicionFinita">
    <simbolo>
        <simbolo>S</simbolo>
    </simbolo>
    <estado-inicio>
        <denominacion>4</denominacion>
    </estado-inicio>
    <estado-fin>
        <denominacion>5</denominacion>
    </estado-fin>
    <denominacion>f( 4, S) = 5</denominacion>
</transicion>
```

Figura 4: Fragmento de archivo XPDL generado por SIAR

El lenguaje XPDL “da soporte a la definición y a la importación / exportación de procesos, con el objetivo de que, aunque se modele un proceso en una aplicación, este modelo pueda ser usado por otras aplicaciones de modelado y/o por otras aplicaciones que trabajen en el entorno de ejecución” [5].

c) Verificar la consistencia secuencial de los escenarios del caso de uso:
Finalmente se ingresa el archivo XPDL, que representa al caso de uso como grafo de estados, al programa “Autómata Finito” que forma parte del conjunto de “Herramientas Prácticas para el Aprendizaje de Informática Teórica”. Esta aplicación java “permite definir autómatas y gramáticas para la enseñanza y ejercitación de los alumnos de la asignatura Sintaxis y Semántica del Lenguaje que se dicta en la carrera de Ingeniería en Sistemas de Información de la Universidad Tecnológica Nacional Facultad Regional Córdoba” [6].

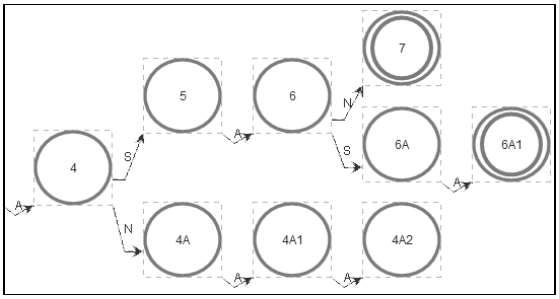

Figura 5: Fragmento del grafo de estados en el simulador de autómatas finitos

El simulador posibilita probar el grafo de estados y comprobar si es aceptado o rechazado. Si es aceptado, significa que la consistencia de los escenarios del caso de uso es correcta. Sino, el rechazo indica que en algunos de los caminos hay un error en la secuencia, pudiéndose visualizar el detalle para detectar la inconsistencia ya sean estados inconexos, inconsistencias o bucles infinitos. Luego, el error puede ser corregido en una nueva versión del caso de uso, generando su correspondiente grafo de estados automáticamente.
De esta manera se logra control y trazabilidad de los cambios de estados en todos los escenarios que conforman un caso de uso, el cual describe un requerimiento funcional.
Vale recordar a F. P. Brooks cuando dijo: “la tarea más importante que el ingeniero de software hace para el cliente es la extracción iterativa y el refinamiento de los requerimientos del producto” [7].

Resultados

El trabajo puesto en consideración en este artículo se originó dentro del proyecto de investigación “Validación de

Requerimientos a través de Modelos Conceptuales" del GIDTSI (Grupo de Investigación, Desarrollo y Transferencia de Sistemas de Información), dependiente del Departamento Ingeniería en Sistemas de Información de la Universidad Tecnológica Nacional, Facultad Regional Córdoba.

El proyecto "busca dar solución a uno de los principales problemas de la Ingeniería en Sistemas relacionado a la elicitación y especificación de requerimientos, que vincula las distintas etapas del proceso de desarrollo de software manteniendo la trazabilidad de los mismos hasta su validación e implementación" [8].

Para alcanzar esta meta, se han analizado y comprendido el procedimiento y el modelo de datos asistido por una aplicación web denominada SIAR que gestiona los requerimientos funcionales de un sistema de información según los lineamientos de UML y permite implementar un método automatizado para validar la cohesión de un caso de uso desde el punto de vista de la transición de estados definidos intrínsecamente en los pasos de su especificación funcional. Haciendo posible también enlazar este proyecto con un trabajo académico de la carrera Ingeniería en Sistemas de Información, del Grupo de Herramientas Didácticas de Informática Teórica que contribuyó con el simulador de autómatas finitos. La decisión de desarrollar una herramienta propia y no utilizar una de las ya existentes que soportan modelado de casos de uso en UML, fue tomada porque a futuro nos permite ampliar sus funcionalidades integrando otros programas y/o metodologías de acuerdo a los avances del proyecto y a las necesidades de comprobación de nuestra investigación.

Discusión

Así como es aceptado universalmente en la actualidad el uso de UML para el modelado de sistemas, también son reconocidos los errores comunes en la especificación de requerimientos utilizando casos de uso. Según J.A. Pow-Sang: "Lamentablemente, la bibliografía existente, muestra muchas formas de aplicar los casos de uso y no son pocas las veces en que su empleo es inadecuado. Algunas de las causas son: mala interpretación del estándar UML y secuencia incorrecta de actividades para la creación de casos de uso." [9].

Además "una especificación incorrecta de requerimientos puede ocasionar errores ocultos dentro del sistema" [10] que repercutan en efecto dominó sobre las fases posteriores del desarrollo de un sistema, provocando lo que Mizuno [11] denominó una "catarata de errores".

Aunque ya se sabe que ninguna metodología es infalible, no implica que por ello no sea perfectible. Por ejemplo, Brooks promueve hacer foco en resolver la complejidad esencial del proceso de software, no perdiendo tiempo en las complejidades accidentales, para lograr importantes mejoras, por él estimadas "mayor que diez veces en un período de diez años" [7]. Coincidiendo en esta línea de pensamiento, el proyecto de investigación que enmarca este trabajo, se ocupa de la *validación y trazabilidad de los requerimientos* de un sistema de información al momento de formular su Modelo Conceptual.

Se identificó como una de las validaciones factibles de implementar la consistencia de los escenarios definidos en un caso de uso. Para ello se reutilizó el conocimiento y el desarrollo informático de otro proyecto de la Facultad logrando un resultado positivo en tiempo y forma. La justificación de que el mencionado desarrollo esté basado en autómatas finitos deterministas es parte de las conclusiones de un trabajo anterior donde se expone que las propiedades de validación de estas máquinas abstractas "son consideradas completas en la validación de los mismos, ya que el único punto que no está presente en comparación de otras herramientas de model Checker, es el control de deadlock, o bloqueo mutuo, el cual no es necesario considerarlo ya que por la particularidad de la elección dentro de las máquinas Abstractas, como son los Autómatas Finitos, su definición,

configuración y forma de funcionamiento no permiten que esta situación se plantee ya que las entradas son secuenciales y procesadas únicamente con desplazamiento en un solo sentido." [12].

Sin embargo, es pertinente la observación de por qué no se investigó en líneas paralelas y complementarias software derivado de "Model Checking" [13]; ya que este mismo planteo se hizo el grupo de trabajo en forma interna y se lo incluyó como uno de los próximos alcances del proyecto. Motivó la decisión el trayecto requerido para abordar esta técnica en forma análoga, como ser el proceso propuesto por Shinkawa [14] para verificación de modelos formales de sistemas, y poder agregar a SIAR una salida adicional en lenguaje de ProMeLa (Proceso o Protocolo Meta Lenguaje) y una interfase con el "model checker" SPIN.

Otra limitación de alcance de este análisis que corresponde aclarar, es que aplica solamente en forma individual para cada caso de uso y no se revisa la consistencia "entre" los casos de uso de un mismo Modelo Conceptual.

Resumiendo, dado que "el proceso de desarrollo de software es aquel en que las necesidades del usuario son traducidas en requerimientos de software, estos requerimientos transformados en diseño y el diseño implementado en código, el código es probado, documentado y certificado para su uso operativo. Concretamente define quién está haciendo qué, cuándo hacerlo y cómo alcanzar un cierto objetivo." [1], la validación del diseño es de primordial importancia para la construcción e implementación de un sistema. Por ello, se presenta este método de análisis simple y concreto, en pos de sumar un eslabón más a la validación del Modelo Conceptual que formaliza el diseño de un software, pues como dijo Edsger W. Dijkstra: "La simplicidad es un prerrequisito para la fiabilidad".

Conclusión

A la pregunta planteada en la Introducción *¿Cómo se definiría un método, basado en simuladores de autómatas finitos deterministas, que verifique la consistencia de los escenarios definidos en un caso de uso?*, se responde con la puesta a consideración este método automatizado, basado en el aplicativo SIAR que registra, normaliza y transforma un caso de uso a formato XPDL y un simulador de autómatas finitos, que permite verificar la consistencia secuencial de los distintos caminos del caso de uso.

El aporte que realiza SIAR al proyecto en el que fue concebido, es el de constituirse como plataforma de software integradora de las aplicaciones que se utilizan en cada una de las líneas de investigación.

Agradecimientos

A las respectivas familias, por su apoyo a nuestra labor académica, y a los colegas investigadores del grupo GIDTSI de U.T.N. F.R.C..

Referencias

[1] J. Rumbaugh, I. Jacobson, G.Booch. "The Unified Modelling Language Reference". Addisson Wesley 1999.

[2] Samarjit Chakraborty. "Formal Languages and Automata Theory-Regular Expressions and Finite Automata-". Computer Engineering and Networks Laboratory Swiss Federal Institute of Technology (ETH) Zurich. Marzo 2003.

[3] Ivar Jacobson y otros.Object Oriented Software Engineering. A Use Case Driven Approach. Addison Wesley, 1992.

[4] C. Leonardi, J.C.S. Leite, Gustavo Rossi. "Una estrategia de Modelado Conceptual de Objetos, basada en Modelos de requisitos en lenguaje natural". Tesis de Maestría Universidad Nacional de la Plata Año 2001.

[5] J. D. Pérez. "Notaciones y lenguajes de procesos. Una visión global". Tesis de Doctorado Universidad de Sevilla.

[6] Proyecto Construcción de Herramientas Didácticas para la enseñanza y ejercitación práctica en laboratorio de Informática Teórica en las Carrreras con Informática. "Manual de Usuario – Grupo de Herramientas Didácticas". Sitio web:http://www.profesores.frc.utn.edu.ar/sistemas/ssl/Marciszack/GHD/Main.htm.U.T.N. – F.R.C. 2009.

[7] Frederick P. Brooks. "No Silver Bullet. Essence and Accidents in Software Engineering". IEEE Computer. Abril 1987.

[8] Marcelo Marciszack, Ramiro Pérez y Claudia Castro. "Validación de Requerimientos a través de

Modelos Conceptuales – Modelos y Transformaciones". WICC 2013.
[9] Pow-Sang, J.A., La Especificación de Requisitos con Casos de Uso: Buenas y Malas Prácticas, II Simposio Internacional de Sistemas de Información e Ing. de Software en la Sociedad del Conocimiento-SISOFT 2003, Pontificia Universidad Católica del Perú, Lima-Perú, 200.
[10] Antonelli, Leandro. "Traceability en la elicitación y especificación de requerimientos". Tesis de Magister en Ingeniería de Software. Facultad de Informática. Universidad Nacional de La Plata. La Plata, Febrero 2003.
[11] Mizuno Y., "Software Quality Improvement", IEEE Computer, Vol. 16, No. 3, Marzo 1983.
[12] Manuel Perez Cota, Mario Groppo, Marcelo Marciszack. "Validación de Especificaciones Funcionales en el modelado de Esquemas Conceptuales a través de Máquinas Abstractas". CoNaIISI 2013. 1er. Congreso Nacional de Ingeniería Informática / Sistemas de Información. Córdoba, Argentina. Noviembre 2013.
[13] Edmund Clarke, Allen Emerson, and Joseph Sifakis. Model Checking: Algorithmic Verification and Debugging. ACM 2007 Turing Award.
[14] Yoshiyuki Shinkawa. "Model Checking for UML Use Cases". Faculty of Science and Technology, Ryukoku University. 2008.

Datos de Contacto:

Marcelo Martín Marciszack, marciszack@gmail.com – Oscar Carlos Medina, oscarcmedina@gmail.com – Claudia Castro, ingclaudiacastro@gmail.com – Enrique Humberto Moyano, enriquemoyano@gmail.com.
GIDTSI, Grupo de Investigación, Desarrollo y Transferencia de Sistemas de Información - Universidad Tecnológica Nacional - Facultad Regional Córdoba –Maestro López esq. Av. Cruz Roja Argentina, Ciudad Universitaria – (5016) Córdoba.