XVII Jornadas de Ingeniería del Software y Bases de Datos

Sistedes 2012

ACTAS

JISBD | PROLE | JCIS

Sistedes
Sociedad de Ingeniería de Software y Tecnologías de Desarrollo de Software

IN LUMINE SAPIENTIA · UNIVERSITAS ALMERIENSIS ·

Almería, 17 al 19 de Septiembre

Editores: Antonio Ruíz | Luis Iribarne

A. Ruíz, L. Iribarne (Eds.): Actas de las "XV*II Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2012)*", Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.

JISBD 2012

XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD)

Almería, 17 al 19 de Septiembre de 2012

Editores:
Antonio Ruíz
Luis Iribarne

Actas de las "*XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD)*"
Almería, 17 al 19 de Septiembre de 2012
Editores: Antonio Ruíz y Luis Iribarne
http://sistedes2012.ual.es
http://www.sistedes.es

ISBN: 978-84-15487-28-9
Depósito Legal: AL 674-2012

Prólogo

Las XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD) (JISBD 2012) se celebraron del 17 al 19 de Septiembre de 2012 en Almería y fueron organizadas por Grupo de Investigación de Informática Aplicada de la Universidad de Almería. Al igual que en anteriores ediciones, JISBD se celebró en paralelo y compartiendo algunos actos de las XII Jornadas de Programación y Lenguajes (PROLE) y de las VIII Jornadas de Ciencia e Ingeniería de Servicios (JCIS). Lo tres eventos son organizados bajo el auspicio de SISTEDES, la Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software.

JISBD se ha consolidado como un foro de referencia donde investigadores y profesionales de España, Portugal e Iberoamérica, en los campos de la Ingeniería del Software y de las Bases de Datos, pueden debatir e intercambiar ideas, crear sinergias y, sobre todo, conocer la investigación que se está llevando a cabo en dicha comunidad. A fin de conseguir de manera efectiva este espacio de intercambio, las jornadas se organizaron por sesiones temáticas en las que han tenido cabida hasta cinco tipos de contribuciones: (1) trabajos regulares, que presentan algún resultado de investigación, (2) trabajos emergentes, que están comenzando su andadura, (3) demostraciones de herramientas, (4) trabajos relevantes ya publicados y (5) tutoriales. Para iniciar el debate indicando los aspectos más destacables y los más discutibles de cada contribución, los coordinadores de sesión delegaron parcialmente dicha responsabilidad en la figura del contraponente de cada contribución.

Las sesiones temáticas de esta edición han sido:

- *Sesión 1:* Bases de Datos, Almacenes de Datos, Minería de Datos, Recuperación de la información
- *Sesión 2:* Ingeniería Web, Interfaces de Usuario, Sistemas Colaborativos, Computación Ubicua
- Sesión 3: Apoyo a la decisión en Ingeniería del Software, Metodologías, Experimentación
- *Sesión 4:* Calidad, Pruebas y Requisitos
- *Sesión 5:* Desarrollo de Software Dirigido por Modelos
- *Sesión 6:* Líneas de Producto, Componentes y Arquitecturas Software
- *Sesión 7:* Otros aspectos de Ingeniería del Software y Bases de Datos.

Este volumen presenta las 86 contribuciones que han formado parte de esta edición: 35 trabajos regulares (con un 71% de ratio de aceptación), 19 trabajos emergentes (con un 89% de ratio de aceptación), 18 trabajos ya publicados, 14 herramientas y 2 tutoriales. También ofrece una breve reseña de la charla invitada impartida por el profesor Armando Fox de la Universidad de California, Berkeley titulada: “Cruzando el abismo educativo” de la ingeniería de software utilizando Software como Servicio y computación en nube. Agradezco que aceptara formar parte de estas Jornadas y su más que colaborativa disposición.

Un signo que acompaña la madurez de la comunidad es la existencia de un abanico de herramientas software cada vez más poblado y de mayor calidad. En esta edición se dispuso un comité de apoyo para su revisión y se organizó una breve sesión plenaria el último día donde dar a conocer y discutir sobre el “mapa de herramientas” de la comunidad JISBD. Estamos convencidos de que esta iniciativa aumentará las sinergias entre los grupos de investigación y por ende aumentará el valor del conocimiento científico y tecnológico que va atesorando nuestra comunidad.

Me gustaría expresar mi más sincero agradecimiento a los miembros del Comité de Programa por su tiempo y dedicación a la hora de revisar y seleccionar los artículos que fueron finalmente aceptados para su presentación, y que han permitido confeccionar un año más un programa de gran calidad y nivel. También a los distintos Coordinadores que se han ocupado de organizar aspectos esenciales como las demostraciones de herramientas (Cristina Vicente y Fernando Sánchez), trabajos relevantes (Amador Durán), tutoriales (Ángeles Saavedra) y coordinadores de las diferentes sesiones temáticas. Por supuesto, mi agradecimiento a los autores que enviaron artículos a las Jornadas, hayan sido aceptados o no, por su esfuerzo y contribución al evento.

También me gustaría agradecer al equipo del comité de organización liderado por Luis Iribarne su gran esfuerzo y excelente trabajo, que han permitido hacer realidad esta conferencia; al Comité Permanente de las JISBD por depositar su confianza a la hora de presidir el Comité de Programa, y por su constante apoyo y soporte. Mención especial merece Coral Calero, cuyos consejos y ayuda como presidente saliente han sido siempre inestimables. Un especial agradecimiento a la Universidad de Almería, que ha hecho posible que la conferencia fuera todo un éxito. Asimismo, este evento no hubiera sido posible sin el aval de la Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software (SISTEDES) y sin la colaboración de la Asociación de Técnicos de Informática (ATI), y la oficina española del W3C.

Muchas gracias a todos los asistentes y participantes a las JISBD 2012, y esperamos verles de nuevo en las próximas JISBD.

Almería, Septiembre 2012

Antonio Ruiz-Cortés
Presidente del Comité de Programa de JISBD 2012

Prologo de la Organización

Las jornadas SISTEDES 2012 son un evento científico-técnico nacional de ingeniería y tecnologías del software que se celebra este año en la Universidad de Almería durante los días 17, 18 y 19 de Septiembre de 2012, organizado por el Grupo de Investigación de Informática Aplicada (TIC-211). Las Jornadas SISTEDES 2012 están compuestas por las XVII Jornadas de Ingeniería del Software y de Bases de Datos (JISBD'2012), las XII Jornadas sobre Programación y Lenguajes (PROLE'2012), y la VIII Jornadas de Ciencia e Ingeniería de Servicios (JCIS'2012). Durante tres días, la Universidad de Almería alberga una de las reuniones científico-técnicas de informática más importantes de España, donde se exponen los trabajos de investigación más relevantes del panorama nacional en ingeniería y tecnología del software. Estos trabajos están auspiciados por importantes proyectos de investigación de Ciencia y Tecnología financiados por el Gobierno de España y Gobiernos Regionales, y por proyectos internacionales y proyectos I+D+i privados. Estos encuentros propician el intercambio de ideas entre investigadores procedentes de la universidad y de la empresa, permitiendo la difusión de las investigaciones más recientes en ingeniería y tecnología del software. Como en ediciones anteriores, estas jornadas están auspiciadas por la Asociación de Ingeniería del Software y Tecnologías de Desarrollo de Software (SISTEDES).

Agradecemos a nuestras entidades colaboradoras, Ministerio de Economía y Competitividad (MINECO), Junta de Andalucía, Diputación Provincial de Almería, Ayuntamiento de Almería, Vicerrectorado de Investigación, Vicerrectorado de Tecnologías de la Información (VTIC), Enseñanza Virtual (EVA), Escuela Superior de Ingeniería (ESI/EPS), Almerimatik, ICESA, Parque Científico-Tecnológico de Almería (PITA), IEEE España, Colegio de Ingenieros Informática de Andalucía, Fundación Mediterránea, y a la Universidad de Almería por el soporte facilitado. Asimismo a D. Félix Faura, Director de la Agencia Nacional de Evaluación y Prospectiva (ANEP) de la Secretaría de Estado de I+D+i, Ministerio de Economía y Competitividad, a D. Juan José Moreno, Catedrático de la Universidad Politécnica de Madrid, presidente de la Sociedad de Ingeniería y Tecnologías del Software (SISTEDES), a D. Francisco Ruiz, Catedrático de la Universidad de Castilla-La Mancha, y a D. Miguel Toro, Catedrático de la Universidad de Sevilla, por su participación en la mesa redonda "*La investigación científica informática en España y el año Turing*"; a Armando Fox de la Universidad de Berkley (EEUU) y a Maribel Fernández del King's College London (Reino Unido), como conferenciantes principales de las jornadas, y a los presidentes de las tres jornadas por facilitar la confección de un programa de *Actividades Turing*. Especial agradecimiento a los voluntarios de las jornadas SISTEDES 2012, estudiantes del Grado de Ingeniería Informática y del Postgrado de Doctorado de Informática de la Universidad de Almería, y a todo el equipo del Comité de Organización que han hecho posible con su trabajo la celebración de una nueva edición de las jornadas JISBD'2012, PROLE'2012 y JCIS'2012 (jornadas SISTEDES 2012) en la Universidad de Almería.

Luis Iribarne
Presidente del Comité de Organización
@sistedes2012{JISBD;PROLE;JCIS}

Comité Científico

Presidente del Comité de Programa:

Antonio Ruiz Cortés (Universidad de Sevilla)

Coordinadores de Demostraciones:

Cristina Vicente-Chicote (Univ. Politécnica de Cartagena)
Fernando Sánchez (Univ. Extremadura)

Coordinadora de Tutoriales:

Ángeles Saavedra Places (Univ. A Coruña)

Coordinador de Divulgación de Trabajos Relevantes ya Publicados:

Amador Durán (Univ. de Sevilla)

Coordinadores de Sesiones Temáticas:

Coordinadores Sesión Temática 1:

Alfredo Goñi (Univ. País Vasco)
José Francisco Aldana (Univ. de Málaga).

Coordinadores Sesión Temática 2:

Pascual González (Univ. Castilla-La Mancha)
Juan Carlos Preciado (Univ. Extremadura)

Coordinadores Sesión Temática 3:

Mercedes Ruiz (Univ. Cádiz)
Agustín Yagüe (Univ. Politécnica de Madrid)

Coordinadores Sesión Temática 4:

Xavier Franch (Univ. Politécnica de Catalunya)
Claudio de la Riva (Univ. Oviedo)

Coordinadores Sesión Temática 5:

Antonio Vallecillo (Univ. Málaga)
José Raúl Romero (Univ. Córdoba)

Coordinadores Sesión Temática 6:

Carlos Canal (Univ. Málaga)
Silvia Abrahão (Univ. Politécnica Valencia)

Coordinadores Sesión Temática 7:

Coral Calero (Univ. Castilla-La Mancha)

Comité de Programa:

Ambrosio Toval (Univ. Murcia)
Ana María Moreno (Univ. Polit. Madrid)
Ana Moreira (Univ. Nova Lisboa)
Antonio Polo (Univ. Extremadura)
Antonio Rito (Univ. Tec. Lisboa)
Arantza Illarramendi (Univ. País Vasco)
Arantza Irastorza (Univ. País Vasco)
Artur Boronat (Univ. Leicester)
Carles Farré (Univ. Polit. Catalunya)
Carme Quer (Univ. Polit. Catalunya)
Cristina Cachero (Univ. Alicante)
Daniel Rodríguez (Univ. Alcalá)
David Benavides (Univ. Sevilla)
Dolors Costal (Univ. Polit. Catalunya)
Eduardo Fdez-Medina (Univ. Castilla-La Man)
Emilio Insfrán (Univ. Polit. Valencia)
Ernest Teniente (Univ. Polit. Catalunya)
Ernesto Pimentel (Univ. Málaga)
Esther Guerra (Univ. Autónoma de Madrid)
Félix García (Univ. Castilla-La Mancha)
Francisco Gutiérrez-Vela (Univ. Granada)
Francisco Ruiz (Univ. Castilla-La Mancha)
Goiuria Sagardui (Univ. Mondragón)
Ignacio Panach (Univ. Valencia)
Irene Garrigós (Univ. Alicante)
Isidro Ramos (Univ. Polit. Valencia)
Ismael Sanz (Univ. Jaume I)
Jaime Gómez (Univ. Alicante)
Javier Cámara (Univ. De Coimbra)
Javier Dolado (Univ. País Vasco)
Javier Jaén (Univ. Polit. Valencia)
Javier Tuya (Universidad de Oviedo)
Jenifer Pérez (Univ. Polit. Madrid)
Jesús García Molina (Univ. Murcia)
Jesús Torres (Univ. Sevilla)
Jesús Aguilar (Univ. Pablo Olavide)
Joan Fons (Univ. Polit. Valencia)
Joao Araujo (Univ. Nova Lisboa)
João Falcão e Cunha (Univ. Porto)
Jon Iturrioz (Univ. País Vasco)
Jordi Cabot (École des Mines de Nantes)
José Hilario Canós (Univ. Polit. Valencia)
José Luis Arjona (Univ. Huelva)
José Luis Fernández-Alemán (Univ. Murcia)
José Luis Roda (Univ. La Laguna)
José María Cavero (Univ. Rey Juan Carlos)
José Norberto Mazón (Univ. Alicante)

José Ramón Paramá (Univ. A Coruña)
José Riquelme (Univ. Sevilla)
José Samos (Univ. Granada)
Juan Carlos Trujillo (Univ. Alicante)
Juan de Lara (Univ. Aut. Madrid)
Juan Garbajosa (Univ. Polit. Madrid)
Juan Hernández (Univ. Extremadura)
Juan José Moreno (Univ. Polit. Madrid)
Juan Manuel Murillo (Univ. Extremadura)
Juan Manuel Vara (Univ. Rey Juan Carlos)
Juan Sánchez (Univ. Polit. Valencia)
Luis Iribarne (Univ. Almería)
Mª Esperanza Manso (Univ. Valladolid)
Mª José Escalona (Univ. Sevilla)
Macario Polo (Univ. Castilla-La Mancha)
Manuel Fernández-Bertoa (Univ. Málaga)
Manuel Nuñez (Univ. Comp. de Madrid)
Manuel Resinas (Univ. Sevilla)
Marcela Genero (Univ. Castilla-La Mancha)
María José Aramburu (Univ. Jaume I)
Maribel Sánchez-Segura (U. Carlos III)
Mario Piattini (Univ. Castilla-La Mancha)
Miguel Goulao (Univ. Nova Lisboa)
Miguel R.Luaces (Univ. A Coruña)
Miguel Toro (Univ. Sevilla)
Natalia Juristo (Univ. Polit. Madrid)
Nelly Bencomo
Nieves Brisaboa (Univ. A Coruña)
Orlando Ávila-García (Open Canarias S.L.)
Oscar Díaz (Univ. País Vasco)
Oscar Dieste (Univ. Polit. Madrid)
Oscar Pastor (Univ. Polit. Valencia)
Óscar Pedreira (Univ. A Coruña)
Pablo de la Fuente (Univ. Valladolid)
Patricia Paderewski (Univ. Granada)
Pedro J.Clemente (Univ. Extremadura)
Pedro Pablo Alarcón (Univ. Polit. Madrid)
Pedro Sánchez (Univ. Polit. Cartagena)
Pepe Carsí (Univ. Polit. Valencia)
Rafael Berlanga (Univ. Jaume I)
Rafael Capilla (Univ. Rey Juan Carlos)
Rafael Corchuelo (Univ. Sevilla)
Robert Clarisó (UOC)
Roberto Ruiz (Universidad Pablo Olavide)
Salvador Trujillo (IKERLAN)
Santiago Meliá (Univ. Alicante)
Sergio Segura (Univ. Sevilla)
Sira Vegas (Univ. Polit. Madrid)
Toni Urpí (Univ. Polit. Catalunya)

Valeria De Castro (Univ. Rey Juan Carlos)
Verónica Bollati (Univ. Rey Juan Carlos)
Vicente Luque Centeno (Univ. Carlos III)
Vicente Pelechano (Univ. Polit. Valencia)
Víctor Sánchez (Open Canarias)
Yania Crespo (Univ. Valladolid)

Comité de Organización

Presidente:

Luis Iribarne (Universidad de Almería)

Miembros:

Alfonso Bosch (Universidad de Almería)
Antonio Corral (Universidad de Almería)
Diego Rodríguez (Universidad de Almería)
Elisa Álvarez, Fundación Mediterránea
Javier Criado (Universidad de Almería)
Jesús Almendros (Universidad de Almería)
Jesús Vallecillos (Universidad de Almería)
Joaquín Alonso (Universidad de Almería)
José Andrés Asensio (Universidad de Almería)
José Antonio Piedra (Universidad de Almería)
José Francisco Sobrino (Universidad de Almería)
Juan Francisco Inglés (Universidad Politécnica de Cartagena)
Nicolás Padilla (Universidad de Almería)
Rosa Ayala (Universidad de Almería)
Saturnino Leguizamón (Universidad Tecnológica Nacional, Argentina)

Índice de Contenidos

Sesiones Temáticas

Sesión Temática 1: Bases de Datos, Almacenes de Datos, Minería de Datos, Recuperación de la información.
Coordinadores: Dr. Alfredo Goñi y Dr. José Francisco Aldana

Sesión Temática 2: Ing. Web, Interf. Usuario, Sist. Colaborativos, Computación Ubicua
Coordinadores: Dr. Pascual González y Dr. Juan Carlos Preciado

---

Sesión Temática 3: Apoyo decisión Ing. Software, Metodologías, Experimentación
Coordinadores: Dra. Mercedes Ruiz y Dr. Agustín Yagüe

Sesión Temática 4: Calidad, Pruebas y Requisitos
Coordinadores: Dr. Xavier Franch y Dr. Claudio de la Riva

Sesión Temática 5: *Desarrollo de Software Dirigido por Modelos*
Coordinadores: Dr. Antonio Vallecillo y Dr. José Raul Romero

Sesión Temática 6: Líneas de Producto, Componentes y Arquitecturas Software
Coordinadores: Dr. Carlos Canal y Dr. Silvia Abrahão

Sesión Temática 7: Miscelánea
Coordinadora: Dra. Coral Calero

Charla Invitada

Crossing the Software Education Chasm using Software-as-a-Service and Cloud Computing

Armando Fox

A. Ruíz, L. Iribarne (Eds.): Actas de las "*XVII Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2012)*", Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.

Crossing the Software Education Chasm using Software-as-a-Service and Cloud Computing

Prof. Armando Fox
Computer Science Division, University of California, Berkeley
fox@cs.berkeley.edu

Via the remarkable alignment of cloud computing, software as a service (SaaS), and Agile development, the future of software has been revolutionized in a way that also allows us to teach it more effectively. Over the past 3 years we have been reinventing UC Berkeley's undergraduate software engineering course to cross the long-standing chasm between what many academic courses have traditionally offered and the skills that software employers expect in new hires: enhancing legacy code, working with nontechnical customers, and effective testing. In our course, "two-pizza teams" of 4 to 6 students create a prototype application specified by real customers (primarily nonprofit organizations) and deploy it on the public cloud using the Rails framework and Agile techniques. Students employ user stories and behavior-driven design to reach agreement with the customer and test-driven development to reduce mistakes. During four 2-week iterations, they continuously refine the prototype based on customer feedback, experiencing the entire software lifecycle–requirements gathering, testing, development, deployment, and enhancement–multiple times during a 14-week semester. Because of Rails' first-rate tools for testing and code quality, students learn by doing rather than listening, and instructors can concretely measure student progress. We have also successfully repurposed those same tools to support nontrivial machine grading of complete programming assignments, allowing us to scale the on-campus course from 35 to 115 students and offer a Massively Open Online Course (MOOC) to over 50,000 students. Indeed, to support instructors interested in adopting our techniques in their classes, we provide not only an inexpensive textbook and prerecorded video lectures to complement the curriculum, but also a set of questions and programming assignments that includes free autograding. Our experience has been that students love the course because they learn real-world skills while working with a real customer, instructors love it because students actually practice what they learn rather than listening to lecture and then coding the way they always have, and employers love it because students acquire vital skills missing from previous software engineering courses.

A. Ruíz, L. Iribarne (Eds.): JISBD'2012, pp. 21, ISBN:978-84-15487-28-9.
Jornadas SISTEDES'2012, Almería 17-19 sept. 2012, Universidad de Almería.

Enfoque dirigido por modelos para probar Sistemas de Información con Bases de Datos

Federico Toledo Rodríguez[1] , Beatriz Pérez Lamancha[2], Macario Polo Usaola[3],

[1] Abstracta, Montevideo, Uruguay.
ftoledo@abstracta.com.uy
[2]Centro de Ensayos de Software, Universidad de la República, Montevideo, Uruguay
bperez@fing.edu.uy
[3]Universidad de Castilla-La Mancha, Ciudad Real, España
macario.polo@uclm.es

Abstract. La base de datos es un componente esencial de los sistemas de información. En efecto, para una base de datos dada, una organización tendrá, probablemente, múltiples aplicaciones que la gestionen, de acuerdo a los diferentes tipos de usuarios, plataformas, dispositivos, etcétera. Aunque la propia base de datos ya incorporará su propio conjunto de restricciones, la lógica de los programas asociados a ella debe también contemplarlas y manejarlas correctamente. En este artículo proponemos un enfoque de generación de casos de prueba centrado en la base de datos, donde se prueba el comportamiento de las aplicaciones que la gestionan, comprobando si son capaces de manejar las particularidades de las estructuras definidas en forma adecuada para las distintas capas de la arquitectura (lógica, interfaz de usuario, etc.). Esta propuesta representa el modelo de datos usando *UML Data Modeling Profile*, el cual es extraído automáticamente a partir del esquema relacional de la base de datos mediante técnicas de ingeniería inversa, para luego generar el modelo de pruebas de forma automática usando transformaciones entre modelos. Dicha transformación busca ocurrencias de patrones que, desde el punto de vista del *testing*, representan situaciones de prueba interesantes, y genera casos de prueba siguiendo el estándar *UML Testing Profile*. En este artículo se describe el entorno de trabajo y se presentan, a modo de ejemplo, el diseño de las pruebas para las operaciones de creación y eliminación de instancias.

Keywords: Pruebas de Software, Pruebas basadas en modelos, Pruebas de Sistemas de Información, Datos de Prueba.

1 Introducción

En el desarrollo de software las pruebas juegan un papel muy importante, suponiendo alrededor de la mitad del costo del proyecto. Con el fin de minimizar los costos y aumentar la productividad se apuesta por la automatización de sus tareas, que Meudec [1] clasifica en: (1) tareas administrativas (registro de especificaciones, generación de informes, etc.); (2) tareas mecánicas (ejecución de casos, captura y reejecución, etc.);

y (3) tareas de generación de casos de prueba. Mientras que se han conseguido muchos avances en la automatización de las dos primeras, queda aún mucho trabajo por hacer en cuanto a la generación automática de casos, debido en gran parte a la diversidad de entornos y contextos posibles en los que se ejecutarán los casos: en efecto, Ball et al. [2] citaban en 2000 tres líneas de investigación en este sentido, que siguen aún siendo válidas: (1) generación de casos a partir de código fuente para alcanzar algún nivel de cobertura; (2) generación de casos a partir del conocimiento del sistema que tenga el tester y del comportamiento esperado del sistema; y (3) generación a partir de especificaciones formales.

Los sistemas de información (SI) son uno de esos contextos y entornos que hemos mencionado que condicionan la generación automática de casos. En ellos, la base de datos (BD) desempeña un papel fundamental. Por lo general, una misma BD es "atacada" por aplicaciones diversas en función, por ejemplo, del tipo de usuario, el sistema operativo, el entorno de ejecución, el tipo de dispositivo por el que se accede, etc. En muchos casos, existe una correspondencia clara entre el modelo de la BD y la capa de dominio de las aplicaciones que la acceden: de hecho, hay multitud de trabajos cuyo objetivo principal es la propuesta de alguna técnica de ingeniería inversa de la BD para obtener su esquema conceptual y utilizar éste como modelo de la capa de dominio de nuevas aplicaciones de gestión de los datos [3-5]. El esquema conceptual puede obtenerse en forma de un diagrama de clases de *Unified Modeling Languange* (UML): en este caso, se le pueden aplicar técnicas de transformación de modelos para, por ejemplo, generar automáticamente casos de prueba, que es el principal tema de este artículo.

En la última década la utilización de modelos para las diferentes actividades del desarrollo de software se hace cada vez más común. Las pruebas de software no quedan fuera de esta tendencia, destacándose el *Model-Driven Testing* (MDT), que se refiere a pruebas basadas en modelos donde los casos de prueba son generados automáticamente desde artefactos de software mediante transformación de modelos. En este trabajo proponemos usar un enfoque MDT para generar pruebas automáticamente a partir de los metadatos de la BD.

Así, y puesto que la BD es el principal elemento común entre estas aplicaciones, podemos aprovechar su estructura como punto de partida para la construcción de casos de prueba que permitan garantizar la calidad de los programas de acceso y gestión de los SI.

En este trabajo se propone generar pruebas automáticamente para verificar SI que almacenan datos en BD relacionales. Para esto se trabaja sobre un metamodelo basado en UML (extraído automáticamente por mecanismos de ingeniería inversa) que representa el modelo de datos. Luego, para cada subestructura interesante sobre el modelo de datos se generan casos de prueba automáticamente.

Tras describir en la sección 2 los antecedentes y nuestra motivación, se presenta brevemente el enfoque general que se abordará, indicando los distintos estándares que se utilizarán para lograr generalizar la propuesta a otros tipos de SI, y por último se muestran los primeros avances en la generación de pruebas basados en un modelo de datos obtenido a partir de una BD.

2 Contexto de trabajo y motivación

La motivación principal de trabajar en esta temática surgió del trabajo en la industria, en particular probando SI desarrollados con GeneXus©. GeneXus [6] es una herramienta 4GL que permite el desarrollo con un enfoque dirigido por modelos: a partir de la especificación en alto nivel se genera código para distintas plataformas. Esto ha evolucionado en los últimos 20 años de acuerdo al avance de las tecnologías, pasando de sistemas basados en RPG, Windows, Web, y actualmente también para dispositivos móviles. Esta herramienta, de gran implantación en América Latina, Japón, y Estados Unidos, comienza ahora a implantarse en España, en donde ya cuenta con importantes clientes como entidades bancarias. GeneXus tiene su centro en Logroño, donde se realizan anualmente encuentros de usuarios, el último celebrado en marzo de 2011. A finales de 2011 más de 7.000 empresas usan GeneXus, con un total de más de 85.000 usuarios [7]. Esta herramienta permite, manteniendo el modelo de datos, migrar entre las diversas tecnologías. Por ello, para un mismo modelo de datos, puede ser necesario probar la aplicación generada para un ambiente de escritorio, y luego para un ambiente web. Es notorio que lo más importante aquí es poder probar las capas de la aplicación y presentación al usuario, viendo si estas son capaces de manipular correctamente el modelo de datos. Entonces, es interesante generar pruebas centrándose en el modelo de datos que sean válidas para las distintas plataformas que puedan implementarse sobre el mismo.

Desde el 2005 el Centro de Ensayos de Software (CES, www.ces.com.uy) ha trabajado en pruebas de SI en Uruguay y en la región, en muchos casos sobre sistemas desarrollados con esta tecnología. Dado el volumen de trabajo, se vio la conveniencia de crear una herramienta específica para automatizar pruebas de SI construidos con GeneXus, por lo que se creó la empresa Abstracta (www.abstracta.com.uy) que, desde 2007, fabrica el producto GXtest©, que a día de hoy está extendiéndose por la comunidad de usuarios GeneXus, alcanzando ya clientes en 10 países.

Los SI desarrollados con GeneXus son conformes a un metamodelo propietario, no estándar, al cual denominan *Knowledge Base* (KB), o base de conocimiento. En GeneXus se modelan gráficamente las entidades que maneja el SI, sus atributos, relaciones y, a partir de esto, la herramienta genera el modelo de datos y las capas de aplicación y presentación que manejan estos datos, esto es, el soporte a las operaciones de creación, modificación, lectura, eliminación, y listados, para que el usuario final pueda manipular los datos que se pueden almacenar en ese modelo de datos. Luego, GeneXus permite programar, en un lenguaje de alto nivel, procedimientos y eventos que permiten el procesamiento de los datos.

GXtest permite automatizar pruebas al mismo nivel de abstracción que GeneXus: es decir, asociando los artefactos de prueba a los elementos de la KB. Con las herramientas tradicionales se asocian los artefactos de prueba a la aplicación generada, y el problema mayor con esto es que al regenerar la aplicación (ya sea por un cambio en el sistema, o por un cambio de plataforma en la que se desea generar) los artefactos de prueba generalmente deben ser reconstruidos. Con GXtest, sin embargo, se mantienen asociaciones directas entre los artefactos de prueba y los elementos de la KB, absorbiendo el impacto de los cambios.

Una vez que se contaba con la plataforma de automatización de pruebas para GeneXus, se comenzó a investigar acerca de la generación automática de pruebas en

base a la información obtenida en la KB: o sea, a partir del modelo de datos de la aplicación, se generan pruebas que son automáticamente ejecutables sobre el sistema generado.
Vistos los buenos resultados obtenidos en GXtest en la generación de pruebas automática basada en el modelo de datos para SI generados con GeneXus, se ve la posibilidad de generalizar la metodología desarrollada en GXtest hacia otros tipos de SI. Para esto un buen camino a tomar es enlazando con estándares, aprovechando tanto el trabajo existente en la academia y la industria (ya sea sobre modelado, generación de pruebas, etc.), como nuestra propia experiencia (tanto la adquirida en el CES y en Abstracta, como en el Grupo Alarcos de la Universidad de Castilla-La Mancha).

3 Conceptos Previos

Como veremos con detalle en las secciones siguientes, nuestro enfoque de generación automática de casos de prueba consta de tres etapas principales: (1) extracción del modelo de datos; (2) búsqueda de patrones en el modelo y generación de casos de prueba en forma de modelos; (3) generación de casos de prueba ejecutables a partir de los modelos de prueba, que sirvan para probar el SI.

Con objeto de hacerlo lo más generalista posible, trataremos de utilizar, siempre que sea posible, especificaciones estándares para bregar con la representación de los modelos y las transformaciones. Así, nos basaremos en el uso de UML 2.0, que permite su extensión, entre otros mecanismos, mediante perfiles, que se basan en el uso de estereotipos y de valores etiquetados:

- El *UML Data Modeling Profile* (UDMP) [8] es una extensión de UML propuesta por IBM para diseñar BD en UML, pero manteniendo el poder expresivo del modelo de entidad-relación extendido. Define tanto los conceptos a nivel físico y de arquitectura (*Node*, *Tablespace*, *Database*, etc.), hasta los más útiles para el diseño de BD (*Table*, *Column*, etc.). Algunos otros trabajos utilizan también este perfil para el modelado de BD [9-11].
- El *UML Testing Profile* (UTP) [12] es un perfil definido por OMG, que define un lenguaje para diseñar, visualizar, especificar, analizar, construir y documentar artefactos de prueba. Extiende UML con conceptos específicos para las pruebas, agrupándolos en: arquitectura de pruebas, datos de pruebas, comportamiento de pruebas y tiempos de prueba. La arquitectura de las pruebas contiene la definición de todos los conceptos necesarios para realizar las pruebas. El comportamiento de las pruebas especifica las acciones y evaluaciones necesarias para la prueba. El caso de prueba es el concepto principal en el modelo de prueba, y su comportamiento se describe usando el concepto *Behavior* de UML 2.0, diagramas de secuencia, máquinas de estado o diagramas de actividad. En este perfil, un caso de prueba (*test case*) es una operación de un contexto de prueba que especifica cómo un conjunto de componentes cooperan con el sistema bajo prueba para alcanzar el objetivo de prueba, retornando un veredicto. Como es un perfil, se integra sin problemas con UML.
- Respecto de las transformaciones entre modelos, OMG también ha adoptado y publicado el lenguaje de transformación entre modelos QVT (*query/view/transformation*) [13], definido a nivel de metamodelado. Usando

QVT es posible lanzar consultas sobre modelos, y por supuesto, realizar transformaciones entre modelos.

- Además, OMG también ha publicado un lenguaje de transformación modelo-a-texto, llamado MOFM2T [14]. El objetivo es definir un lenguaje que facilite la generación de código fuente o documentación textual a partir de modelos.

En nuestro grupo de investigación tenemos ya cierta experiencia en la utilización de los distintos estándares en diferentes contextos, y también en la búsqueda de patrones en modelos de datos (si bien con un objetivo distinto: la generación automática de servicios web [5]). En este trabajo se aprovechará la experiencia previa en el desarrollo de herramientas y técnicas de reingeniería (por ejemplo, [15]), pruebas ([16]), herramientas comerciales de generación de pruebas (GXtest) y transformaciones de modelos con QVT y UTP [17], con el fin de construir un entorno de pruebas genérico para probar SI.

4 Vista General del Proceso para Generación de Casos de Prueba

Nuestro enfoque consta de las tres etapas que se muestran en la **Fig. 1** y que se describen a continuación:

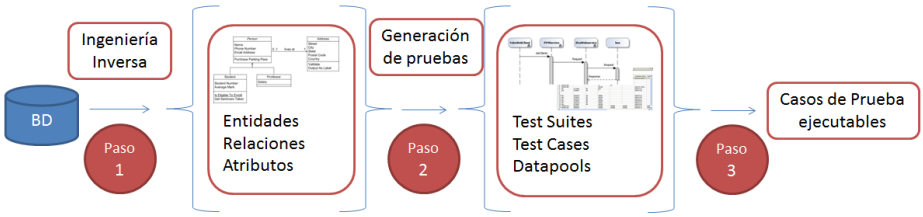


Fig. 1. Esquema general de la metodología

- **Paso 1: Extracción del modelo de datos.** Mediante técnicas y herramientas de ingeniería inversa se obtiene, a partir del esquema físico de la BD, su correspondiente modelo de datos en la forma de un diagrama de clases conforme al UDMP de IBM, en el que quedan representadas las entidades con sus relaciones y atributos. Logramos así representar el modelo de datos en forma independiente de la plataforma sobre la que ejecutará el sistema.
- **Paso 2: Búsqueda de patrones y generación de casos de prueba mediante MDT**. El modelo de datos se procesa mediante técnicas de transformación de modelos. Se buscan, mediante reglas QVT, ciertos patrones interesantes, los cuales no son más que subestructuras del modelo de datos que representan situaciones propicias para ser probadas. Como resultado de las transformaciones, se generan casos de prueba representados con el UTP.
- **Paso 3: Generación de código de prueba.** Por último, tomando como entrada los modelos UTP generados y el modelo de datos, se generan código o scripts de prueba mediante transformaciones MOFM2T.

Respecto del segundo paso, resulta interesante poder dotar al entorno de suficiente capacidad de extensión como para que sea posible definir independientemente nuevos patrones, de manera que se permita incrementar el conocimiento acumulado en generación de pruebas: deseamos que, al ir probando distintos sistemas, se pueda ir enriqueciendo la base de patrones de pruebas, tanto por la propia experiencia y experimentación de este enfoque en sistemas de distinta naturaleza, como de su aplicación a perfiles concretos de usuarios o dominios determinados.

Algo importante a destacar del último paso es que los casos de prueba ejecutables se generan con invocaciones a métodos de una capa de adaptación también generada automáticamente, que luego deberán ser implementados para que accedan a las operaciones correspondientes y sean completamente ejecutables. Esto permite entonces tener pruebas ejecutables siguiendo un enfoque de *Keyword-driven testing* [18]. La mayor ventaja es que se pueden utilizar las mismas pruebas para ejecutarlas contra la aplicación generada para diferentes plataformas.

Hasta el momento tenemos resuelto por completo el primer paso, y estamos trabajando en el segundo y el tercero. Para el primero se trabaja principalmente con las herramientas *Enterprise Architect* (EA, www.sparxsystems.com/products/) y Papyrus en Eclipse (www.eclipse.org/papyrus/), para el modelado de UML. La herramienta IBM Rational Rose Data Modeler, de la compañía que propone el perfil de datos, paradójicamente no lo utiliza, sino que tiene un formato propietario. La ventaja de EA es que da soporte, también gráfico, al UDMP. También hemos extendiendo *Relational Web*, la cual es una herramienta de reingeniería desarrollada previamente en nuestro grupo, adaptándola para obtener el modelo de datos automáticamente a partir del esquema de una BD y guardarlo en formato UDMP siguiendo las técnicas presentadas por Polo et al. [15]. Para el resto del proceso se está trabajando en el diseño del método de generación de pruebas, que deberá garantizar el alcance de ciertos criterios de cobertura sobre el modelo de datos, de lo que nos ocupamos en la siguiente sección.

5 Diseño de Casos de Prueba a partir de Modelos de Datos

Los casos de prueba se generan mediante transformaciones realizadas sobre el modelo de datos. En éste, se identifican situaciones de prueba interesantes (obviamente, desde el punto de vista del testing) mediante la búsqueda de subestructuras del modelo que concuerden con patrones de modelos predefinidos (una situación de prueba interesante, por ejemplo, podría ser la existencia de una asociación *1:** entre dos clases, que podría proceder de la ingeniería inversa de una relación *1:n* entre dos tablas: quizás sea interesante probar el funcionamiento del SI ante la inserción de un registro en la tabla secundaria sin un registro asociado en la tabla principal).

Es importante destacar que las pruebas generadas atacan el sistema completo, basándose en las estructuras de la BD, por lo que se está verificando que el sistema bajo pruebas sea capaz de manejarlas correctamente. Si por ejemplo en el diseño de pruebas un caso de prueba intenta usar un dato que no cumple una regla CHECK que indica que cierto campo entero debe ser mayor que un valor dado (por ejemplo, el campo *Edad* debe ser mayor que 18), claramente esto será bien manejado por el gestor de BD, pero es muy importante ver que el sistema reaccione adecuadamente ante la ex-

cepción que le dará la ejecución de la sentencia SQL, y que el modelo de datos de la BD continúe consistente con el modelo de datos de las capas superiores.

Toda estrategia de generación de pruebas debe ser mensurable. En nuestro caso, también debemos ser capaces de generar casos de prueba que alcancen algún nivel determinado de cobertura, de manera que pueda cuantificarse su completitud. Los criterios son un mecanismo para comprobar, validar y cuantificar que lo que estamos probando realmente "cubre" el modelo, de acuerdo a una teoría de errores dada. A modo de prueba de concepto, nos basaremos en un criterio de cobertura en particular, pero siempre teniendo en cuenta que el entorno no se limita sólo a los criterios aquí planteados, sino que es extensible a nuevos criterios o casos de prueba que se deseen utilizar más adelante.

5.1 Criterios de Cobertura para Diagramas de Clases

En primer lugar, de entre los criterios de cobertura para modelos UML, propuestos por Andrews et al. [19], consideraremos el subconjunto propuesto para diagramas de clases. Puesto que en este trabajo se propone generar los casos de prueba a partir de diagramas de clase, tiene sentido que se mida la completitud de los casos de prueba mediante los siguientes criterios:

- **AEM (*Association-end multiplicity*):** el conjunto de pruebas debe causar que se cree cada par de multiplicidades representativo en las asociaciones del modelo. Así, si existe una asociación cuya multiplicidad es, en un extremo, *p..n*, debería instanciarse la asociación con *p* elementos (valor mínimo), *n* elementos (valor máximo) y con uno o más valores en el intervalo *(p+1, n-1)*.
- **GN (*Generalization*):** el conjunto de pruebas debe conseguir que se cree cada relación de generalización del modelo. Si bien el modelo relacional (del cual procede el diagrama de clases) no posee herencia, seguimos el criterio de Premerlani y Blaha [20], quienes consideran como tales aquellas relaciones de clave externa con cardinalidades *1* a *0:1* entre claves principales.
- **CA (*Class attribute*):** el conjunto de pruebas debe conseguir que se creen conjuntos de valores representativos de los diferentes atributos de cada clase. El conjunto de valores representativos se consigue en tres pasos: (1) crear valores representativos para cada atributo, para lo que puede usarse la técnica de clases de equivalencia; (2) calcular el producto cartesiano de estos valores; (3) eliminar los conjuntos de valores inválidos de acuerdo con el dominio del problema, posibles restricciones que anoten el diagrama, etc.

5.2 Criterios de Cobertura *CRUD*

Además de los criterios anteriores, y dado que todo dato del SI tiene un ciclo de vida que viene determinado por las operaciones que afectan a su persistencia (operaciones *CRUD: Create*, *Read*, *Update*, *Delete)*, también tiene sentido prestar atención a la pruebas de estas operaciones. El ciclo de vida de un dato viene dado por una ocurrencia de una expresión regular del tipo $C{\cdot}R{\cdot}(U_i{\cdot}R)^*{\cdot}D{\cdot}R$ [21], en donde cada U_i representa una de las múltiples operaciones de actualización del objeto. Las operaciones *R (Read)* son simples operaciones de lectura que no alteran el estado del objeto ni el de

sus datos persistentes asociados, por lo que se utiliza para construir el oráculo tras cada operación que sí suponga cambio de estado. Así, por ejemplo, para una *Factura*, un posible ciclo de vida del que excluimos la operación *Delete* sería *crear factura, asignar cliente, escribir fecha, añadir línea 1, añadir línea 2, modificar línea 1, añadir descuento, calcular IVA, calcular total*, en donde todas las operaciones (salvo la primera) son instancias de alguna $Update_i$. Entre cada par de operaciones se añadiría una operación *Read* para comprobar que el objeto se actualiza correctamente.

En trabajos previos [22] construimos una herramienta que genera casos de prueba expandiendo expresiones regulares y combinando con datos de prueba, por lo que sería interesante verificar la efectividad de cruzar ambos enfoques. Sin embargo, la expansión de la expresión regular y su posterior combinación con datos de prueba puede dar lugar a un número casi inmanejable de casos de prueba. Para limitar este número utilizaremos los criterios de cobertura para expresiones regulares de [23], que ya hemos empleado también en otras ocasiones [24].

5.3 Ejemplos de Aplicación

Para ilustrar la aplicación de los criterios de cobertura anteriores, presentaremos el análisis de cómo generar casos de prueba para alcanzar el criterio AEM de los propuestos por Andrews et al., y, luego, de las operaciones interesantes para las entidades de un SI (operaciones CRUD), nos concentramos en los casos particulares de *create* y *delete*.

Para una asociación entre dos tablas, para alcanzar el criterio AEM es necesario considerar las multiplicidades de la relación. Cuando de BD se trata, las relaciones entre dos tablas pueden ser simples (máximo es 1) o múltiples (máximo es N). Las relaciones pueden ser obligatorias u opcionales, en función de si la multiplicidad mínima es 0 o 1. De aquí surgen 16 situaciones que resulta interesante considerar aparte de cada asociación del modelo de datos, las cuales son las combinaciones de las cuatro posibilidades en cada uno de los dos extremos:

- 0..1 (simple y opcional)
- 1 (simple y obligatoria)
- 0..* (múltiple y opcional)
- 1..* (múltiple y obligatoria)

Para cada una de estas situaciones queremos generar casos de prueba y, para el criterio AEM, es necesario probar con los extremos de los rangos de las multiplicidades. Esto nos hace considerar que los valores interesantes son 0, 1 y *, donde * se entiende como una cantidad *N* mayor a 1. Para el caso de relaciones de multiplicidad "muchos", vemos suficiente con considerar un *N* igual a 2, pues es el mínimo valor con el cual se logran establecer múltiples instancias de una tabla asociadas a la de otra (no obstante, se podrían especificar otros valores de *N* en el momento de generar los casos de prueba). Entonces, instanciaremos las relaciones considerando en los extremos multiplicidades de 0, 1 y 2, lo que da un total de 9 situaciones (combinando las 3 posibilidades para cada uno de los 2 extremos). Cada una de estas las queremos aplicar a los 16 casos.

En la **Tabla 1** se muestran los casos diseñados para cubrir el criterio AEM para la operación *Create* para un caso particular, en el que la tabla **A** se asocia con la tabla **B** por medio de una relación [0..* : 1] (**Fig. 2**). Los patrones de prueba generarán conjuntos de prueba de acuerdo a las particularidades de la relación:

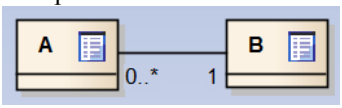
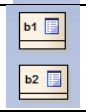
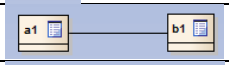
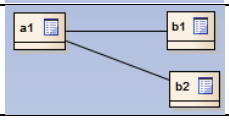


Fig. 2. Ejemplo de relación entre dos tablas

La primera columna de la **Tabla 1** identifica la situación; la segunda y la tercera indican las multiplicidades que se desean crear (o sea, el caso que se está cubriendo), que se muestran gráficamente en la cuarta columna; luego, en las columnas *Resultado Esperado* y *Acción de verificación* se muestra respectivamente el resultado esperado y qué verificar para constatar que la operación fue correcta. Luego, en la **Tabla 2**, se hace el mismo análisis para *Delete*. Para facilitar el análisis se consideraron los 9 casos, y se analizaron los resultados esperados de acuerdo a si cada uno de los extremos es múltiple o no, y si es obligatorio o no. Se parte considerando que no existe ninguna de las instancias a crear (en estos ejemplos no se verifica la unicidad: si se nombran dos instancias diferentes, se supone que no viola ninguna restricción).

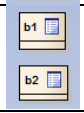
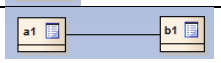
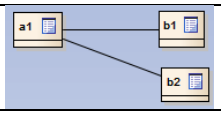
Tabla 1. Diseño de pruebas para la operación *Create*

ID	#A	#B	CREAR	Resultado esperado	Acción de validación
C01	0	0	N/A (no se crea nada)	N/A	N/A
C02	0	1		A no es obligatoria => PASS	Verificar que se creó b1
C03	0	2		A no es obligatoria => PASS	Verificar que se crearon b1 y b2
C04	1	0		B es obligatoria => FAIL	Verificar que no se creó a1
C05	1	1		PASS	Verificar que se creó a1, asociado a b1
C06	1	2		B no es múltiple => FAIL	Verificar que no se pueda asociar a dos elementos
…	…	…	…	…	…

Con objeto de minimizar los tiempos de generación, ejecución y mantenimiento de casos de prueba es interesante mantener, en el *test suite,* el menor número de casos de prueba que, sin embargo, cubran todas estas situaciones interesantes. Además de utilizar, en un futuro, técnicas de reducción de *test suites* como las que hemos aplicado en [25], en nuestro contexto se pueden aprovechar ciertos estados en los que quede la aplicación tras la ejecución de un caso de prueba para probar otro. Por ejemplo, luego de crear un elemento, probar actualizarlo y borrarlo. Entonces, siguiendo este análisis, logramos generar un conjunto de casos de prueba que nos garantizan alcanzar el cu-

brimiento deseado, mientras que minimizamos la ejecución de pruebas gracias al orden de ejecución. Por ejemplo, el resultado de C02 genera un estado en la BD que sirve de entrada para probar D02, así como C05 deja un estado inicial necesario tanto para D05 como D06. Con un breve análisis es fácil ordenar las pruebas de forma tal que los estados iniciales requeridos por algunas pruebas sean generados por otras, determinando así las suites de prueba.

Tabla 2. Diseño de pruebas para la operación *Delete*

ID	#A	#B	ELIMINAR	Resultado esperado	Acción de validación
D01	0	0	N/A (no se borra nada)	N/A	N/A
D02	0	1		A no es obligatoria => PASS	Verificar que se borró b1
D03	0	2		A no es obligatoria => PASS	Verificar que se borraron b1 y b2
D04	1	0		B es obligatoria => N/A	N/A
D05	1	1		Borrar A => PASS	Verificar que se borra a1, y que b1 no tiene nada asociado
D06				Borrar B => FAIL (B es obligatoria)	Verificar que no se borra b1, y sigue asociado a a1
D07	1	2		B no es múltiple => N/A	N/A
…	…	…	…	…	…

Hasta aquí las pruebas están definidas en un alto nivel de abstracción. Luego, a pesar de que no esté explicado en este artículo, al generar el código de pruebas se generarán datos concretos para cada una de las columnas de cada una de las tablas involucradas, respetando los tipos de datos y las restricciones que estén definidas en el modelo de datos (como por ejemplo las reglas *CHECK*).

5.4 Obtención de Casos de Prueba para el UML Testing Profile

Por último, y para comprender la forma en que estos casos de prueba son generados y representados de acuerdo a UTP, se muestra en la **Fig. 3**, como resultado de las transformaciones, los casos de prueba dentro de la arquitectura de pruebas, la que incluye entre otros:

- Un *Test Context*, conteniendo como métodos los casos de prueba (*test cases*) generados;
- Un *Test Component*, responsable de la inicialización de los casos de prueba y de la interacción con el *System Under Test* (*SUT*);

- Un *datapool* por cada entidad probada; cada *datapool* (conjunto de datos de prueba) tiene un *data selector* por cada *test case* para proveer datos específicos a cada uno.

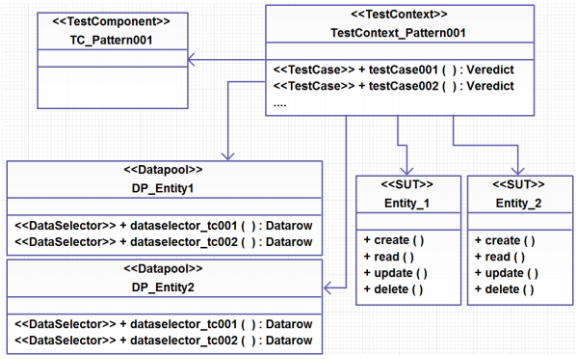


Fig. 3. Arquitectura de pruebas generada en UTP

6 Trabajos Relacionados

Existen varios trabajos que generan datos para sistemas con BD, pero ninguno toma el enfoque dirigido por modelos considerando las operaciones básicas de un SI como proponemos hacer nosotros.

Algunos, por ejemplo, extienden la cobertura basada en líneas de código considerando las particularidades de la interacción del sistema bajo prueba con la BD [26-28], así como las distintas condiciones que se puedan plantear sobre las SQLs que se ejecutan [29, 30]. Así, se generan las instancias de la BD necesarias para cubrir estas situaciones. Tanto en [31] como en [32] se plantea especificar de alguna forma los resultados de las SQL implicadas en la prueba, para de esa forma generar datos para poblar la BD. La propuesta planteada en [33] toma como entradas el esquema de la BD y datos de prueba categorizados dados por el usuario, con lo que genera casos de prueba y estados iniciales para la BD, validando tanto las salidas del sistema como el estado final de la BD. En [34] se generan estados de la BD de acuerdo a las restricciones de integridad del esquema relacional: primero se generan reglas de consistencia de acuerdo a las claves y referencias foráneas, y luego se generan datos que cumplan con ellas.

Existen muchos trabajos que generan pruebas automáticamente a partir de modelos UML [35, 36], pero hasta donde conocemos, solo [37] considera el caso especial de los SI con BD (los cuales tienen particularidades muy importantes a ser tenidas en cuenta). En esa propuesta, Fujiwara et al. proponen generar pruebas considerando un diagrama de clases para representar el modelo de datos, y otro para representar las pantallas. Las relaciones entre tablas son representadas con restricciones OCL, así como también las pre y post condiciones de los métodos a probar. Todo el modelo que especifica el sistema debe ser proporcionado manualmente, y por lo tanto luego debe ser mantenido. Las pruebas que generan están centradas en las restricciones dadas, mientras que en nuestra propuesta veremos que prestamos especial atención al

modelo de datos, y la especificación del sistema es obtenida automáticamente, sin costos de mantenimiento.

7 Conclusiones

En este trabajo se presentó un nuevo enfoque para probar SI que manejan BD, donde se pone especial atención en el cubrimiento de todas las estructuras (simples o compuestas) encontradas en el modelo de datos. Este criterio de adecuación de pruebas tiene en cuenta que en estos sistemas lo importante es que la información esté correctamente almacenada, por lo que se prueban las operaciones que operan sobre estas estructuras conforme a diversos criterios de cobertura.

De esta forma vemos que el enfoque aplicado actualmente a GeneXus, y que es la idea que fue el embrión de este trabajo de investigación (Sección 2), puede ser extendido y aún mejorado para otros tipos de sistemas, generalizando la idea mediante el uso de estándares de la industria, tales como UML, UTP, UDMP, QVT, MOFM2T, etc. Como el entorno de generación de pruebas está casi completamente basado en estándares, puede ser adoptado con casi cualquier herramienta "UML-compliant". Por este motivo, casi no se necesitará implementación de nuevas herramientas para dar soporte a esta metodología.

El artículo ha presentado nuestras primeras ideas y resultados hacia la construcción de un entorno para la generación de pruebas para SI, que irá evolucionando hasta permitir su validación empírica. Por este motivo es importante prestarle atención al trabajo futuro tanto como a las conclusiones presentadas.

8 Trabajo Futuro

A futuro existen muchas líneas de trabajo por donde se plantea continuar.

Por ejemplo, es interesante poder modelar no sólo la estructura del modelo de datos, sino también reglas de negocio que aplican sobre estas estructuras, obtenidas a partir del código fuente, reglas *check*, *deletes* o *updates* en cascada, de los datos del sistema, como para enriquecer las pruebas que de ahí se puedan generar.

Para que las pruebas generadas se puedan ejecutar, como ya se explicó, es necesario indicar (solamente) cómo se accede a cada una de estas operaciones. En el caso de que se esté trabajando en un enfoque de desarrollo dirigido por modelos (MDD) las pruebas podrían ser automáticamente generadas 100%.

Por otro lado, deseamos ir transfiriendo los resultados parciales que se vayan consiguiendo hacia la herramienta GeneXus, lo que nos permitirá, gracias a su amplia implantación, validar los resultados que vayamos obteniendo.

Agradecimientos. Este trabajo ha sido parcialmente financiado por la Agencia Nacional de Investigación e Innovación (ANII, Uruguay), por el proyecto DIMITRI (Desarrollo e Implantación de Metodologías y Tecnologías de Testing, TRA2009_0131) y por el proyecto MAGO/Pegaso (Mejora Avanzada de Procesos Software Globales, TIN2009-13718-C0201).

Referencias

1. Meudec, C., *ATGen: automatic test data generation using constraint logic programming and symbolic execution†.* Software Testing, Verification and Reliability, 2001. **11**(2): p. 81-96.
2. Ball, T., et al., *State generation and automated class testing.* Software Testing Verification and Reliability, 2000. **10**(3): p. 149-170.
3. Alalfi, M.H., J.R. Cordy, and T.R. Dean, *SQL2XMI: Reverse Engineering of UML-ER Diagrams from Relational Database Schemas*, in *Working Conference on Reverse Engineering*. 2008, IEEE Computer Society. p. 187-191.
4. Blaha, M. *A retrospective on industrial database reverse engineering projects. 1*. 2001: IEEE.
5. García Rodríguez de Guzmán, I., *Pressweb: un proceso para la reingeniería de sistemas heredados hacia servicios web*. 2007, UCLM.
6. Artech. *GeneXus*. 1988; Available from: http://www.genexus.com
7. Artech. *GeneXus Facts*. 2012; Available from: www.genexus.com/files/genexus-facts-sheet?es.
8. Gornik, D., *UML Data Modeling Profile*. 2002, IBM, Rational Software.
9. Yin, S. and I. Ray, *Relational database operations modeling with UML*, in *AINA'05: Advanced Information Networking and Applications*. 2005. p. 927-932 vol.1.
10. Zieliriski, K. and T. Szmuc, *Data modeling with UML 2.0.* Software engineering: evolution and emerging technologies, 2006. **130**: p. 63.
11. Sparks, G., *Database modeling in UML*, in *Methods & Tools*. 2001. p. 10-22.
12. OMG. *UML 2.0 Testing Profile Specification*. 2004; Available from: http://utp.omg.org/.
13. OMG, *Meta Object Facility 2.0 Query/View/Transformation Specification*. 2005.
14. OMG, *MOF Model to Text Transformation Language (MOFM2T), 1.0*. 2008.
15. Polo, M., I. García-Rodríguez, and M. Piattini, *An MDA-based approach for database re-engineering.* Journal of Software Maintenance and Evolution: Research and Practice, 2007. **19**(6): p. 383-417.
16. Pérez Lamancha, B. and M. Polo Usaola, *Testing product generation in software product lines using pairwise for features coverage.* Testing Software and Systems, 2010: p. 111-125.
17. Pérez Lamancha, B., et al., *Model-driven Testing - Transformations from Test Models to Test Code*, in *ENASE*. 2011, SciTePress. p. 121-130.
18. Fewster, M. and D. Graham, *Software test automation: effective use of test execution tools*. 1999: ACM Press/Addison-Wesley Publishing Co.
19. Andrews, A., et al., *Test adequacy criteria for UML design models.* Software Testing, Verification and Reliability, 2003. **13**(2): p. 95-127.
20. Premerlani, W.J. and M.R. Blaha. *An approach for reverse engineering of relational databases*. 1993: IEEE.
21. Koomen, T., et al., *TMap Next, for result-driven testing*. 2006: UTN Publishers.
22. Polo, M., S. Tendero, and M. Piattini, *Integrating techniques and tools for testing automation.* Software Testing Verification and Reliability, 2007. **17**(1): p. 3-39.
23. Mariani, L., M. Pezze, and D. Willmor, *Generation of integration tests for self-testing components.* Applying Formal Methods: Testing, Performance, and M/E-Commerce, 2004: p. 337-350.
24. Flores, A. and M. Polo, *Testing-based process for component substitutability.* Software Testing, Verification and Reliability, 2010.
25. Polo Usaola, M., P. Reales Mateo, and B. Pérez Lamancha, *Reduction of Test Suites Using Mutation.* Fundamental Approaches to Software Engineering, 2012: p. 425-438.
26. Haller, K., *White-box testing for database-driven applications: A requirements analysis*. 2009, ACM. p. 13.

27. Shelar, S. and Sdsawarkar, *Database instances generation tool for white-box testing*. 2009. p. 112-116.
28. Emmi, M., R. Majumdar, and K. Sen, *Dynamic test input generation for database applications*, in *ISSTA'07: Software Testing and Analysis*. 2007. p. 151-162.
29. Tuya, J., M.J. Suárez-Cabal, and C. De La Riva, *Full predicate coverage for testing SQL database queries.* Software Testing Verification and Reliability, 2010. **20**(3): p. 237-288.
30. Suárez-Cabal, M.J., C. De La Riva, and J. Tuya, *Populating test databases for testing SQL queries.* IEEE Latin America Transactions, 2010. **8**(2): p. 164-171.
31. Binnig, C., D. Kossmann, and E. Lo, *Multi-RQP - generating test databases for the functional testing of OLTP applications*, in *DBtest'08: International Workshop on Testing Database Systems*. 2008. p. 5.
32. Arasu, A., R. Kaushik, and J. Li, *Data generation using declarative constraints*, in *International conference on Management of data*. 2011, ACM. p. 685-696.
33. Chays, D. and Y. Deng, *Demonstration of AGENDA tool set for testing relational database applications*. 2003, IEEE Computer Society. p. 802-803.
34. Neufeld, A., G. Moerkotte, and P.C. Loekemann, *Generating consistent test data: Restricting the search space by a generator formula.* The VLDB Journal, 1993. **2**(2): p. 173-213.
35. Brucker, A., et al., *A specification-based test case generation method for UML/OCL.* Models in Software Engineering, 2011: p. 334-348.
36. Offutt, J. and A. Abdurazik, *Generating tests from UML specifications.* «UML»'99—The Unified Modeling Language, 1999: p. 76-76.
37. Fujiwara, S., et al., *Test data generation for web application using a UML class diagram with OCL constraints.* Innovations in Systems and Software Engineering, 2011: p. 1-8.