PATROCINADORES

ugr Universidad de Granada

ATC

Sun microsystems

IBM

Bull

Ayuntamiento de Granada

atc

cibersur

AERFAI
Asociación Española para la Inteligencia Artificial.

AERFAI
Asociación Española de Reconocimiento de Formas y Análisis de Imágenes.

AIPO http://grihow.uji.es/AIPO/aipo
Asociación Interacción Persona-Ordenador.

EUROGRAPHICS http://www.eg.org
Capítulo Español de la European Association for Computer Graphics.

EUSFLAT http://www.eusflat.org/index.htm
European Society for Fuzzy Logic and Technology.

SC of the IEEE CIS
Capítulo Español de la IEEE Computational Intelligence Society.

RADISC
Red Andaluza en Sistemas Complejos.

SEPLN http://www.sepln.org
Sociedad Española para el Procesamiento del Lenguaje Natural.

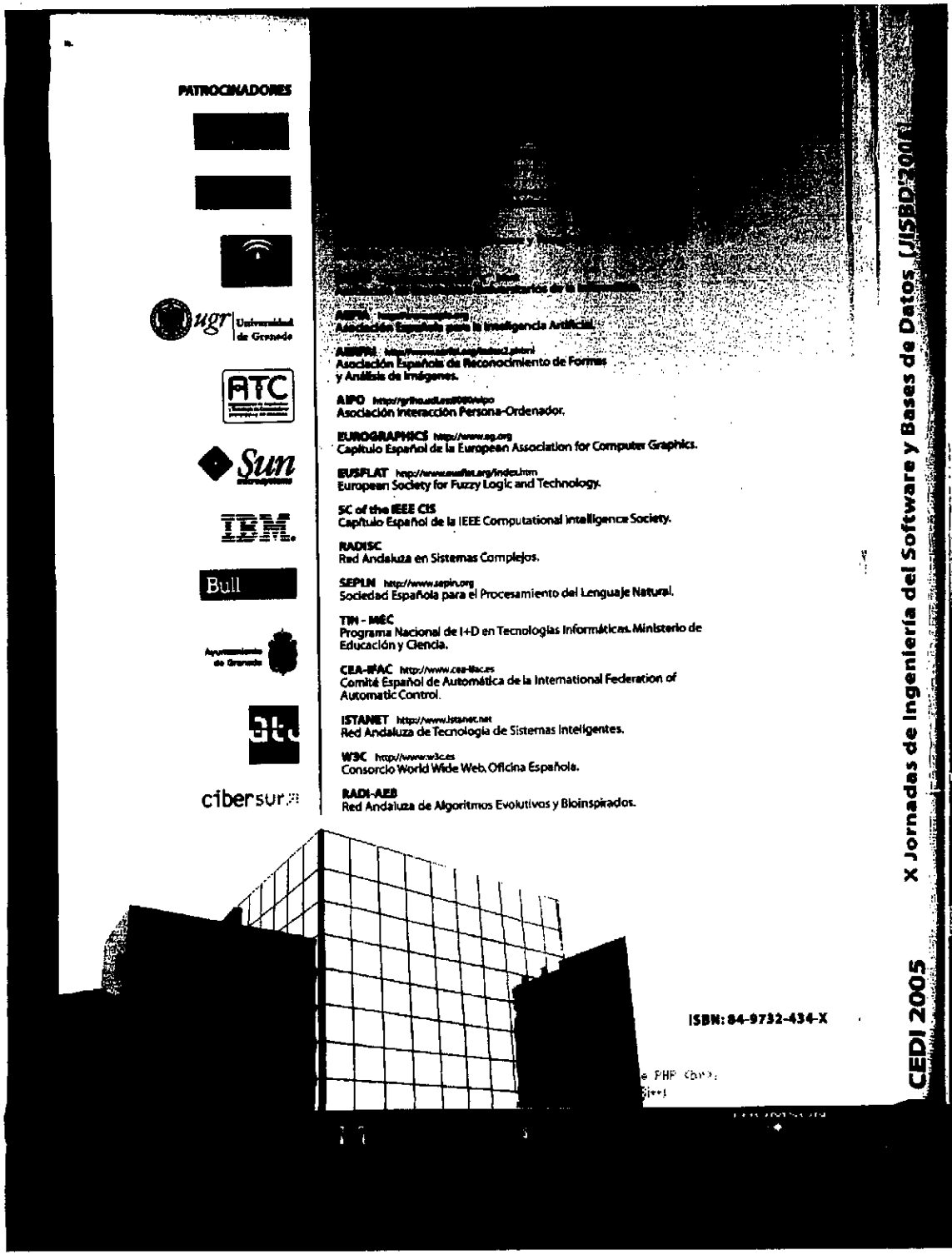
TIN - MEC
Programa Nacional de I+D en Tecnologías Informáticas. Ministerio de Educación y Ciencia.

CEA-IFAC http://www.cea-ifac.es
Comité Español de Automática de la International Federation of Automatic Control.

ISTANET http://www.istanet.net
Red Andaluza de Tecnología de Sistemas Inteligentes.

W3C http://www.w3c.es
Consorcio World Wide Web. Oficina Española.

RADI-AEB
Red Andaluza de Algoritmos Evolutivos y Bioinspirados.



ISBN: 84-9732-434-X

CEDI 2005 X Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2005)

THOMSON

CEDI 2005

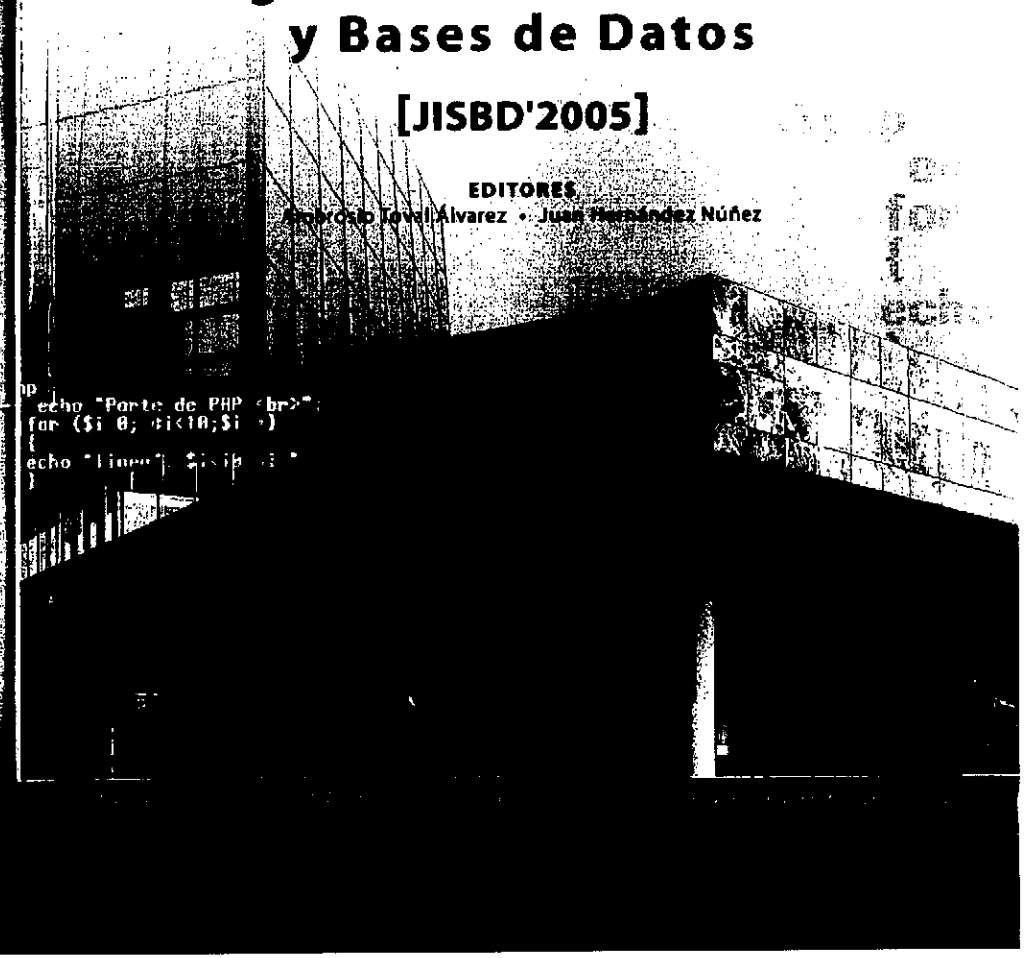
I CONGRESO ESPAÑOL DE INFORMÁTICA

GRANADA DEL 13 AL 16 DE SEPTIEMBRE

X Jornadas de Ingeniería del Software y Bases de Datos

[JISBD'2005]

EDITORES
Arcadio Toval Álvarez • Juan Hernández Núñez



I CONGRESO ESPAÑOL DE INFORMÁTICA

CEDI 2005

Nuevos retos científicos y tecnológicos en Ingeniería Informática

ACTAS DE LAS

X Jornadas de Ingeniería del Software y Bases de Datos

[JISBD'2005]

EDITORES
Ambrosio Toval Álvarez
Juan Hernández Núñez

JORNADAS ORGANIZADAS POR
Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software

THOMSON

Actas de las X Jornadas de Ingeniería
del Software y Bases de Datos [JISBD'2005]

Editores de la serie de Actas del CEDI
Rafael Molina Soriano
Antonio Díaz García
Alberto Prieto Espinosa

Editores de las Actas de las presentes Jornadas
Ambrosio Toval Álvarez
Juan Hernández Núñez

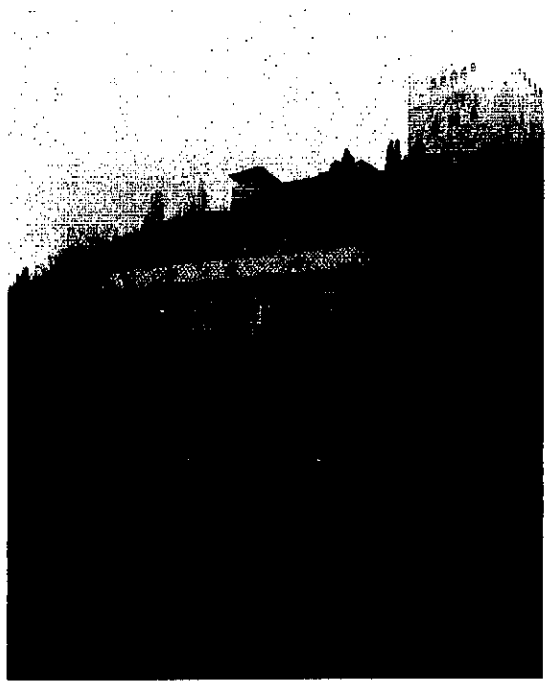
Diseño de Cubiertas
www.dixi-e.com

Impresión
THOMSON

COPYRIGHT© 2005 International
Thomson Editores Spain
Paraninfo, S.A.
Magallanes 25 · 28015 Madrid España
Tel: 91 446 33 50 · Fax: 91 445 62 18
clientes@paraninfo.es

Impreso en España
Printed in Spain

ISBN: 84-9732-434-X
Depósito legal: SE-4046-2005 European Union
Printed by Publidisa



Antiguo Hotel Reuma. Granada
Foto realizada para el CEDI2005 por DIXI

X JORNADAS DE INGENIERÍA DEL SOFTWARE Y BASES DE DATOS

COMITÉ EJECUTIVO

Presidente Comité de Programa	Ambrosio TOVAL (Universidad de Murcia)
Secretario Comisión Permanente	Mario PIATTINI (Universidad de Castilla-La Mancha)
Organización y Relaciones con CEDI'2005	Buenaventura CLARES (Universidad de Granada) Patricia PADEREWSKI (Universidad de Granada)
Coordinador de Tutoriales	José L. FERNÁNDEZ (Universidad de Murcia)
Coordinadora de Talleres	Patricia PADEREWSKI (Universidad de Granada)
Coordinador de Demostraciones	Francisco L. GUTIÉRREZ (Universidad de Granada)

PRESIDENCIA DEL COMITÉ PROGRAMA

Ambrosio TOVAL — Universidad de Murcia

COMITÉ DE PROGRAMA

J. ALDANA (U. Malaga)
B. ÁLVAREZ (U. Politécnica Cartagena)
J. ARAUJO (U. Nova de Lisboa)
M. J. ARAMBURU (U. Castellon)
O. BELO (U. do Minho)
P. BOTELLA (U. Politécnica Cataluña)
N. BRISABOA (U. de La Coruña)
C. CALERO (Castilla-La Mancha)
C. CANAL (U. de Málaga)
J.M. CAVERO (U. Rey Juan Carlos)
M. CELMA (U. P.Valencia)
R. CORCHUELO (U. de Sevilla)
Y. CRESPO (U. de Valladolid)
C. DELGADO (U. Carlos III)
O. DÍAZ (U. Politécnica País Vasco)
J. FALCÃO e CUNHA (U. do Porto)
X. FRANCH (U. Politécnica Cataluña)
P. de la FUENTE (U. de Valladolid)
L. FUENTES (U. de Málaga)
M. J. GASPAR da Silva (U. de Lisboa)
J. GÓMEZ (U. de Alicante)
J. HERNÁNDEZ (U. de Extremadura)
J. ITURRIOZ (U. Politécnica País Vasco)
N. JURISTO (U. Politénica de Madrid)
A. LOPES (U. de Lisboa)
H. MADEIRA (U. de Coimbra)
E. MARCOS (U. Rey Juan Carlos)
J.M. MARQUÉS (U. de Valladolid)
E. MENA (U. Zaragoza)
A. M. MORENO (U. Politénica de Madrid)
J. J. MORENO (U. Politécnica Madrid)
J. M. MURILLO (U. de Extremadura)
N.J. NUNES (U. Madeira)
O. PASTOR (U. Politécnica Valencia)
E. PIMENTEL (U. de Málaga)
A.POLO (U. Extremadura)
C. RAMOS (U. do Algarve)
I. RAMOS (U. Politécnica Valencia)
J. RIQUELME (U. Sevilla)
A. RITO (U. Técnica de Lisboa)
M.J. RODRÍGUEZ (U. Granada)
F. RUIZ (Castilla-La Mancha)
J. SAMOS (U. Granada)
F. SÁNCHEZ (U. de Extremadura)
J. SÁNCHEZ (U. Politécnica Valencia)
S. SOUSA BRITO (I. Politécnico Beja)
E. TENIENTE (U. P. Cataluña)
M. TORO (U. de Sevilla)
J. C. TRUJILLO (U. de Alicante)
B. VELA (U. Rey Juan Carlos)
A. VALLECILLO (U. de Málaga)

REVISORES ADICIONALES

Alberto ABELLÓ	Ma. Valeria de CASTRO
Alfredo GOÑI	Manuel RESINAS
Andre L. SANTOS	María del Mar RODÁN
Ángel HERRANZ	María Luisa RODRÍGUEZ
Antonio Cesar GÓMEZ	María-Isabel SÁNCHEZ-SEGURA
Antonio RUIZ	Marta RUIZ
Arantza ILLARRAMENDI	Miguel Ángel LAGUNA
Artur BORONAT	Nathalie MORENO
Bruno MARTINS	Nelson MEDINILLA
Carlos E. CUESTA	Noelia MAYA
Carme QUER	Norberto FERNÁNDEZ
Daniel GOMES	Nuria MEDINA
Enric MAYOL	Oscar DIESTE
Fran J. RUIZ-BERTOL	Pablo FERNÁNDEZ
Francisco COUTO	Paloma CÁCERES
Francisco Luis GUTIÉRREZ	Patricia PADEREWSKI
Ismael Navas DELGADO	Patricio LETELIER
Javier MUÑOZ	Pedro J. CLEMENTE
Jennifer PÉREZ	Pedro J. MUÑOZ
Joan Antoni PASTOR	Pedro SÁNCHEZ
Joao Pedro NETO	Pedro VALDERAS
José Luis GARRIDO	Rafael BERLANGA
José Miguel BLANCO	Raul ROMERO
José Miguel CAÑETE	Sira VEGAS
José Ramon RIOS	Toni URPÍ
Juan Ángel PASTOR	Vicente LUQUE
Juan Manuel VARA	Vicente PELECHANO
Luis SÁNCHEZ	Xavier FERRÉ

COORDINADOR DE TALLERES

Patricia PADEREWSKI	Universidad de Granada

MIEMBROS COMITÉ DE TALLERES

Mª Visitación HURTADO	Universidad de Granada
Juan Manuel MURILLO	Universidad de Extremadura
José SÁEZ	Universidad de Murcia
Mª Dolores LOZANO	Universidad de Castilla La Mancha
José Hilario CANÓS	Universidad de Valencia
Cecilia DELGADO	Universidad de Granada
Amador DURÁN	Universidad de Sevilla

COORDINADOR DE TUTORIALES

José L. FERNÁNDEZ	Universidad de Murcia

COORDINADOR DE DEMOSTRACIONES

Francisco L. GUTIÉRREZ	Universidad de Granada

SISTEMA AUTOMÁTICO DE REVISIÓN

Quercus Software Engineering Group

Pablo AMAYA	Universidad de Extremadura
Jose M. CONEJERO	Universidad de Extremadura

CONTENIDOS

Artículos.....**1**

Um Quadro de Referência para a Comparação de Metodologias Ágeis.....3
Joao Carlos Ribeiro, Joao Araujo

Búsqueda Tabú para la generación de casos de prueba de cobertura de bucles.11
María Eugenia Díaz Fernández, Raquel Blanco, Javier Tuya

Providing platforms for developing pervasive systems with MDA. An OSGi metamodel.....19
Javier Muñoz, Vicente Pelechano, Estefanía Serral

PRISMANET middleware: Soporte a la Evolución Dinámica de Arquitecturas Software Orientadas a Aspectos.....27
Cristóbal Costa Soria, Jennifer Pérez, Nour Ali, José Á. Carsí, Isidro Ramos

Sistematizando la Especificación de Requisitos Safety: un Caso de Estudio sobre Aplicaciones Teleoperadas.....35
Elena Navarro, Pedro Sánchez, Patricio Letelier, Juan A. Pastor, Isidro Ramos

Una Arquitectura para la Integración de Portales Web basada en Servicios Web Semánticos.....43
César J. Acuña, Juan M. Gómez, Esperanza Marcos, Christoph Bussler

Producción Científica en Ingeniería de Requisitos en España: Un Análisis en el Contexto Europeo.....51
Oscar Dieste, Natalia Juristo, Ana M. Moreno, Alan M. Davis, Ann Hickey

Soporte de Métricas con Independencia del Lenguaje para la Inferencia de Refactorizaciones.....59
Raúl Marticorena Sánchez, Yania Crespo González-Carvajal, Carlos López Nozal

Supporting the Automatic Generation of Advanced Modelling Environments with Graph Transformation Techniques.....67
Esther Guerra, Paloma Díaz, Juan de Lara

Un servicio web de políticas de acceso basadas en roles para hipermedia.....75
Daniel Sanz García, Ignacio Aedo, Paloma Díaz

Síntesis de patrones de interacción a partir de diagramas de secuencia en UML.....83
Miguel Ángel Pérez, Amparo Navasa Martínez, Juan Manuel Murillo, Carlos Canal Velasco

Modelos estructurales de aspectos para arquitectura de software.....91
Carlos E. Cuesta, M. Pilar Romay, Pablo de la Fuente, Manuel Barrio Solórzano

Finding where to apply object-relational database schema refactorings: an ontology-guided approach.....99
Coral Calero Muñoz, Aline Baroni, Fernando Brito e Abreu

Do composite states improve the understanding of UML statechart diagrams?.....107
José Antonio Cruz Lemus, Marcela Genero, Esperanza Manso, Mario Piattini

Transformaciones MDA sobre especificaciones computacionales de UML 2.0 a Maude.....115
José Raúl Romero Salguero, Nathalie Moreno, Antonio Vallecillo

Improving automatic SQL translation for ROLAP tools.....123
Oscar Romero Moral, Alberto Abelló Gamazo

A Hybrid Method for Discovering Distance-Enhanced Inter-Transactional Rules.....131
Pedro Gabriel Ferreira, Ronnie Alves, Paulo Azevedo, Orlando Belo

The Effect of Coupling on Understanding and Modifying OCL Expressions: An Experimental Analysis.....139
Luis Reynoso, Marcela Genero, Mario Piattini, Esperanza Manso

Generación Automática de Aplicaciones Mixtas Sw/Hw mediante la Integración de Componentes COTS.....147
Cristina Vicente Chicote, Ana Toledo Moreo, Carlos Fernández Andrés, Pedro Sánchez

Método de unión de modelos independientes de plataforma en MDA.....155
Álvaro Prieto Ramos, Adolfo Lozano-Tello, Encarna Sosa Sánchez

A product-line approach to database reporting.....163
Felipe I. Anfurrutia, Oscar Diaz, Salvador Trujillo

Un Enfoque Orientado a Procesos para la Especificación de Planes de Emergencia.....171
Manuel Llavador, Patricio Letelier, Marcos R. S. Borges, José H. Canós, Mª Carmen Penadés, Carlos Solís

De la Arquitectura Software al Urbanismo Software: Hacia Nuevas Formas de concebir los Sistemas de Software Intensivo.....179
Juan José Moreno-Navarro

Adaptación de las normas ISO/IEC 12207:2002 e ISO/IEC 15504:2003 para la evaluación de la madurez de procesos software en países en desarrollo.....187
Francisco J. Pino, Felix Garcia, Francisco Ruiz, Mario Piattini

Un entorno integrado para la reingeniería.....195
Ignacio García Rodríguez de Guzmán, Macario Polo Usaola, Mario Piattini

PWSSEC: Proceso de Desarrollo para Seguridad de Servicios.....203
Carlos Gutiérrez García, Eduardo Fernández-Medina, Mario Piattini

Medidas de Usabilidad de Componentes Software.....211
Manuel F. Bertoa, Antonio Vallecillo

ORCDB: Arquitectura para la extensión de la semántica de SQL en bases de datos restrictivas orientadas a objetos con restricciones polinómicas de igualdad.....221
M. Teresa Gómez-López, Rafael M. Gasca, Carmelo Del Valle, Victor Cejudo

Determinación de los requerimientos de calidad del producto software basados en normas internacionales.....231
Abraham Eliseo Dávila Ramón, Karin Ana Melendez Llave, Luis Alberto Flores García

Artículos Cortos.....239
Una aproximación metodológica para soportar la evolución de requisitos a partir de un modelo arquitectónico OA.....241
Amparo Navasa Martínez, Miguel Ángel Pérez, Juan Manuel Murillo

Mejorando la accesibilidad de las aplicaciones GIS basadas en Web.....247
Miguel R. Luaces, Nieves R. Brisaboa, Jose R. Parama, David Trillo, Jose R. R. Viqueira

Del método formal a la aplicación industrial en Gestión de Modelos: Maude aplicado a Eclipse Modeling Framework.....253
Artur Boronat, José Iborra, José Á. Carsí, Isidro Ramos, Abel Gómez

Análisis de los Métodos de Selección de Componentes COTS desde una Perspectiva Ágil.....259
Fredy Javier Navarrete Ramirez, Pere Botella, Xavier Franch

Un Profile para el Modelado de Patrones de Software.....265
José Luis Isla Montes, Francisco Luis Gutiérrez Vela, Patricia Paderewski Rodríguez

Recuperación del conocimiento basada en contexto: Una aplicación en la Arqueología (ArqueOnto).....271
Juan María Fernández González, Antonio Polo Márquez, Luis Jesús Arévalo Rosado, Enrique Cerrillo Cuenca

Desarrollando aplicaciones hipermedia para la Web Semántica.....277
Laura Montells Higuero, Susana Montero, Paloma Díaz, Ignacio Aedo

Arquitectura para la Clasificación y Composición de Servicios Web.....283
Ismael Navas Delgado, Maria del Mar Rojano-Muñoz, Jose F. Aldana-Montes

Diagramas de casos de uso para el análisis de requisitos en almacenes de datos.....289
Jose Norberto Mazón López, Juan Trujillo, Manuel Serrano, Mario Piattini

Especificación de jerarquías de dimensión en un almacén de datos usando WordNet.....295
Jose Norberto Mazón López, Juan Trujillo, Manuel Serrano, Mario Piattini

Un entorno integrado para la reingeniería

Ignacio García-Rodríguez de Guzmán
Dept. de Informática, Estadística y Telemática
Universidad Rey Juan Carlos
28933 Móstoles (Madrid)
Ignacio.Garcia@urjc.es

Macario Polo
Dept. de Informática
Escuela Superior de Informática
Univ. de Castilla-La Mancha
13071 Ciudad Real
Macario.Polo@uclm.es

Mario Piattini
Dept. de Informática
Escuela Superior de Informática
Univ. de Castilla-La Mancha
13071 Ciudad Real
Mario.Piattini@uclm.es

Resumen

Este artículo describe la arquitectura de una herramienta que genera aplicaciones de tres capas (siguiendo la arquitectura multicapa) a partir de bases de datos relacionales. Para ello, la herramienta implementa un completo proceso de reingeniería en el que se utilizan diferentes metamodelos que nos ayudan a representar la información extraída para, posteriormente, generar de manera automática una aplicación que gestione la base de datos. El paso del esquema físico de la base de datos a la aplicación multicapa orientada a objetos se realiza mediante una serie de reestructuraciones y transformaciones en las que los metamodelos empleados juegan un papel fundamental. La herramienta desarrollada, además de permitirnos partir de cuatro gestores de bases de datos distintos, genera aplicaciones con un conjunto muy amplio de funcionalidades para varias plataformas.

Palabras clave: reingeniería, ingeniería inversa, metamodelos.

1. Introducción

El objetivo de la ingeniería inversa es obtener la representación de un sistema software en un mayor nivel de abstracción que el original [3]. Actualmente, la ingeniería inversa se sigue utilizando para recuperar modelos conceptuales y arquitectónicos de sistemas heredados (*legacy*), que son luego utilizados como la base para la fase de ingeniería directa, que produce una nueva versión del sistema adaptado a otros entornos y paradigmas, como el orientado a objetos [16], la computación distribuida [7], el software basado en componentes [1], etc.

De acuerdo con [2], el proceso completo de ingeniería inversa y directa se corresponde con la idea de "reingeniería". La reingeniería puede entenderse como la composición de un conjunto de funciones de transformación que operan sobre diferentes conjuntos, incrementando o decrementando el nivel de abstracción de un producto software. Adicionalmente, puede haber algunas fases intermedias de reestructuración, que cambien algunos de los productos software sin salir del nivel de abstracción.

Por lo general el conjunto inicial comienza desde el código fuente de los programas, siendo los modelos de clases u otros tipos de diagramas los que forman el conjunto final de esta primera transformación. No obstante, ha habido también muchos trabajos que han aplicado ingeniería inversa a otros tipos de productos, como las bases de datos. El objetivo de muchos de ellos es obtener un diagrama que represente el posible esquema conceptual usado durante el desarrollo inicial de la base de datos, con lo que el producto obtenido de la ingeniería inversa suele ser un diagrama entidad-relación (ER) o entidad-relación extendido (EER) [8, 17].

Con esta herramienta, pretendemos generar automáticamente aplicaciones capaces de gestionar bases de datos relacionales con un propósito general (aunque muy fácilmente adaptables a propósitos específicos), por lo que la obtención de un diagrama de clases que exprese la misma semántica que el diagrama ER o EER, nos facilitará esta tarea.

Mediante un diagrama de clases podemos expresar lo mismo que mediante un diagrama ER o EER, y además obtenemos la ventaja de disponer de una representación más cercana a la aplicación objetivo, con lo que la generación de código se hace mucho más sencilla. De hecho, este diagrama de clases también es un punto de partida excelente para generar nuevas versiones de la base de datos.

Cuando se está reconstruyendo un sistema de gestión de información utilizando el paradigma de la orientación a objetos, el modelo de clases correspondiente a la capa de dominio/negocio de la aplicación suele ser utilizado como esquema conceptual para la construcción de la base de datos (en la mayoría de los casos una base de datos relacional [15]), de modo que aproximemos el mundo de los objetos al mundo de las bases de datos. En general, existen al menos tres formas de traducir un conjunto de clases a un conjunto de tablas [4]: "*una clase, una tabla*"; "*un camino de herencia, una tabla*", "un árbol de herencia, una tabla".

En los sistemas de tres capas, el diagrama de clases de dominio es también la base para la creación del resto de capas en una aplicación multicapa, como pueden ser las capas de presentación o de persistencia.

En este artículo presentamos una herramienta, desarrollada en Java, que integra un proceso completo de reingeniería, incluyendo las fases de ingeniería inversa y directa. El punto de partida de la herramienta es una base de datos, sobre cuyo diccionario de datos aplicamos ingeniería inversa para obtener un diagrama de clases independiente de la base de datos; este diagrama es la entrada del proceso automático de generación de código, que construye un sistema multicapa utilizando diferentes lenguajes de programación y plataformas. Además, la aplicación incluye la posibilidad de migrar la base de datos de un gestor a otro. El diseño arquitectónico de la herramienta permite agregar, muy fácilmente, nuevos generadores de código para otros lenguajes. Esta herramienta surge como una evolución del trabajo presentado en [18], una versión muy primitiva de *Relational Web*.

El método está basado en las descripciones matemáticas de los conjuntos y funciones involucradas en el proceso de reingeniería. De acuerdo con Broy [5], la Ingeniería del Software requiere de descripciones matemáticas para sus aspectos de modelado, técnicas de descripción y métodos de desarrollo. Para este autor, esta teoría matemática debería proporcionar significado a las técnicas de descripción habituales (como los diagramas de clases, diagramas de estado, etc.) y debiendo explicar los métodos y algoritmos usando notación matemática, más en la forma de una "Técnica de Descripción Formal", en lugar de cómo un método formal. Las Técnicas de Descripción Formal deberían conducir a una comprensión más profunda de los procesos del software y a un soporte de herramientas mucho más potente.

Este artículo se organiza de la siguiente manera: la sección 2 explica los diferentes pasos de la fase de ingeniería inversa. El proceso de generación de código se presenta y explica en la Sección 3. En la Sección 4 se analizan brevemente algunos entornos relacionados con la reingeniería en las bases de datos. Por último, en la Sección 5 presentamos las conclusiones y las líneas de trabajo futuro.

2. Descripción del proceso de reingeniería

La Figura 1 presenta una vista general del proceso completo implementado por la herramienta: mediante un objeto *Connection*, una fábrica (la clase *DBFactory*) accede al diccionario de datos de la base de datos y construye una instancia de *Database*, una clase que encapsula el metamodelo que representa bases de datos relacionales, sin que exista dependencia con el fabricante. Tras esto, la clase *BDR2OO* aplica un algoritmo de ingeniería inversa a la instancia *Database* para obtener una instancia de *OOS* (*O*bject *O*riented *S*ystem), que es el metamodelo que representa un diagrama de clases UML. El usuario puede manipular el conjunto de clases incluido en la instancia de OOS, añadiendo nuevas operaciones mediante

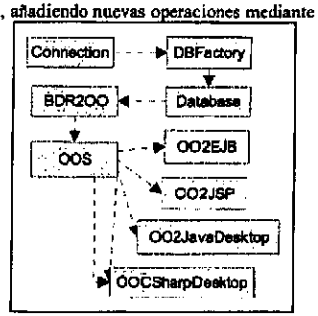


Figura 1. Vista General del proceso de reingeniería

máquinas de estado; tras esto, uno de los generadores de código es utilizado para generar la aplicación de acuerdo a la Tabla 1. En la Sección 3 se describen los algoritmos utilizados para la generación de código.

Generador de código	Capa de presentación	Capa de dominio
OO2EJB	Páginas JSP	Enterprise JavaBeans
OO2JSP	Páginas JSP	Clases Java estándares (estos dos generadores de código comparten las clases de dominio)
OO2JavaDesktop	JFrames, JPanels, JDialogs (de javax.swing)	
OO2CSharpDesktop	Formularios Windows	Clases C#

Tabla 1. Características generales de los generadores de código

2.1. Obtención de una instancia de *Database*

Tal y como muestra la Figura 1, la *DBFactory* accede a la información de la base de datos utilizada como punto de partida del proceso de reingeniería, mediante un objeto *java.sql.Connection*. *Connection* es una interfaz que incluye múltiples operaciones para la recuperación de metadatos de la base de datos. Sin embargo, no todos los gestores de bases de datos implementan todas las operaciones de la interfaz o, en el caso de que lo hagan, la estructura de los resultados de cada operación puede variar de un gestor a otro.

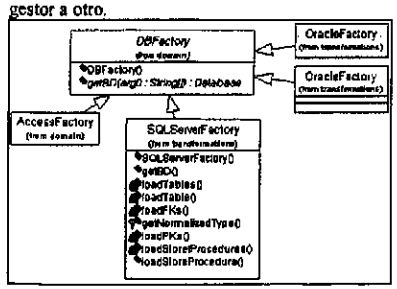


Figura 2. Fábricas para recuperar la estructura de la base de datos

Por ello es necesario utilizar mecanismos diferentes para acceder a la base de datos, dependiendo del gestor de base de datos con el que estemos tratando. Para abordar este problema consideramos la *DBFactory* como la implementación del patrón "*Fabrica abstracta*" [10], que tiene tantas especializaciones como tipos de bases de datos deban ser procesadas: a día de hoy, la herramienta puede manipular bases de datos implementadas en Oracle, Caché InterSystems, Microsoft SQLServer y Microsoft Access, por lo que existen cuatro fabricas concretas (Figura 2).

Database encapsula un metamodelo para representar las estructuras más relevantes de una base de datos relacional, desde el punto de vista de nuestro proceso de reingeniería. Así, *Database* almacena información sobre las tablas y sus columnas, claves ajenas y procedimientos almacenados.

2.2. Fase de ingeniería inversa

La fase de ingeniería inversa es llevada a cabo mediante el elemento *BDR2OO* mostrado en la Figura 1. Su objetivo es obtener un diagrama de clases, que represente el posible esquema conceptual usado durante la construcción de la base de datos.

BDR2OO toma una instancia de la ya descrita *Database* y obtiene una instancia del *OOS* (*Object Oriented System*), que representa el metamodelo de un conjunto genérico de clases interrelacionadas.

Cada clase incluye una máquina de estados, que es añadida a la clase por el usuario de la herramienta, con el fin de otorgarle a las instancias de las clases un comportamiento diferente al que tienen por defecto. Profundizaremos en este aspecto en la Sección 2.4.

Esta definición de "sistema orientado a objetos" es valida para definir cada una de las capas de la aplicación final generada por la herramienta, teniendo en cuenta que cada capa es un subconjunto de la aplicación final orientada a objetos. Suponiendo tres capas en la aplicación final, podríamos describirlo de la siguiente manera:

FinalApp=(PresentationTier, DomainTier, PersistenceTier)

...siendo *PresentationTier*, *DomainTier* y *PersistenceTier* instancias de *OOS*.

Los tres métodos básicos utilizados para traducir un esquema conceptual en una base de datos relacional pueden ser utilizados para llevar a cabo el proceso opuesto: nuestra herramienta obtiene el diagrama de clases asumiendo que se utilizó el patrón "una clase una tabla" durante el desarrollo de la base de datos.

Por defecto, cada clase recibe:

- Un constructor sin parámetros, que construye instancias "vacías", a las cuales asigna valores por defecto en cada campo.
- Un constructor materializador, que es usado para construir instancias de la clase correspondiente de los registros almacenados en la base de datos. Este constructor toma como argumento los valores de la clave primaria correspondientes al registro a materializar.
- Métodos para insertar, actualizar y eliminar instancias de la base de datos.
- Métodos Set/Get para leer y escribir los valores de los campos.

Igualmente se procesa el conjunto de relaciones en el sistema orientado a objetos para agregar métodos que permitan navegar entre las instancias relacionadas, correspondientes a los registros de las tablas relacionadas.

Esta función, que opera sobre el *Database* para producir un *OOS*, es implementada como un método llamado *getOOS* en la clase *BDR2OO* mostrada en la Figura 1.

2.3. Refinamiento del OOS para la generación de código

La Figura 3 muestra la pantalla principal de la herramienta de reingeniería tras aplicar ingeniería inversa a una pequeña base de datos que representa a un banco y utilizada como caso de ejemplo: la pestaña seleccionada (*JSP Application*) contiene un árbol que muestra la estructura del proyecto que será generado; a la derecha, se abrirá una nueva pestaña cada vez que el usuario pulse sobre el nombre de un archivo. En el lado derecho, se ha incluido también una pestaña especial para configurar las opciones de generación de código.

2.4. Adición de maquinas de estado para la definición del comportamiento

Las máquinas de estado pueden ser vinculadas a las clases de dominio para proveerlas de comportamiento y métodos que no sean aquellas que la herramienta les asigna por defecto.

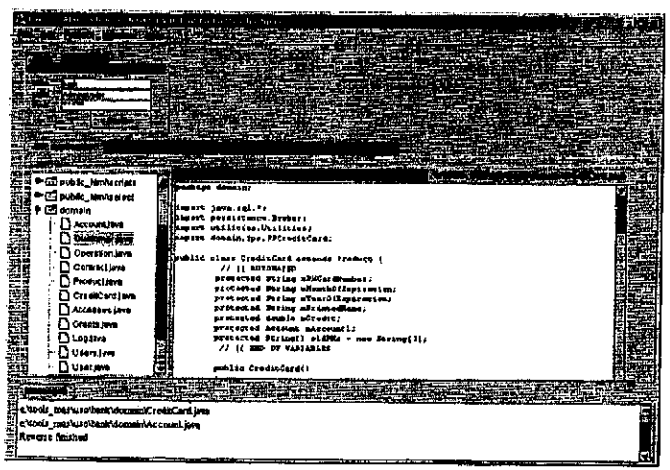


Figura 3. Pantalla principal de la herramienta

La Figura 4 representa la maquina de estados que describe el posible comportamiento de las instancias de la clase *Account* mediante un diagrama de *Rational Rose* (abajo) y mediante nuestra herramienta (arriba). En nuestra herramienta, cada estado recibe:

- Un nombre, como puede ser *BalanceNegative*
- Una descripción, que es una expresión lógica que denota si la instancia está en un estado o no. Por lo general, esta expresión está escrita en función tanto de los campos de la clase como de los métodos observadores. En la parte inferior de la Figura 4, la descripción del estado denota que la cuenta está en el estado *NegativeBalance*, cuando *getBalance()* es menor que cero.
- Un conjunto de transiciones de salida. Cada una tiene un evento disparador, que puede ser una condición (que se evalúa tras ejecutar el evento), una o más acciones y el estado destino. Las transiciones de salida en la parte inferior de la Figura 4 denotan que, antes de ejecutar las acciones del método *deposit(double amount)*, se evalúa una condición; si esta resulta verdadera, entonces la instancia ejecutará la correspondiente acción, situando al objeto en el estado destino.
- Opcionalmente, un conjunto acciones de entrada y salida, que son ejecutadas cada vez que la instancia llega o abandona un estado.

El procesamiento de las clases de dominio con máquinas de estados vinculadas provoca la adición de campos y métodos adicionales a las clases generadas:

- Por cada estado de la máquina de estados, un método que retorna un valor lógico llamado *stateIs_STATE_NAME* (donde *STATE_NAME* corresponde al nombre dado al estado), que devuelve *verdadero* si la expresión utilizada para describir el estado es verdadera, y *falso* en caso contrario.
- Por cada estado de la máquina de estados, un método sin tipo de retorno llamado *setState_STATE_NAME*, cuyo cuerpo incluye un conjunto de acciones de entrada añadidas al estado. Para cada evento (teniendo en cuanta todos los eventos y los estados de la clase), se añade un método nuevo a la clase. La ejecución de este método sólo será posible cuanto el estado de las instancias tenga este evento dentro de su conjunto de transiciones de entrada. Por ejemplo, como puede observarse en la Figura 4, el evento *withdraw* no puede ser ejecutado en el estado *BalanceNegative*. Por ello, el método generado incluye un conjunto de instrucciones condicionales para comprobar si la instancia se encuentra en el estado adecuado.

Para el ejemplo mostrado en la Figura 4, la herramienta genera (además del código por defecto), los campos y métodos mostrados en la Figura 5.

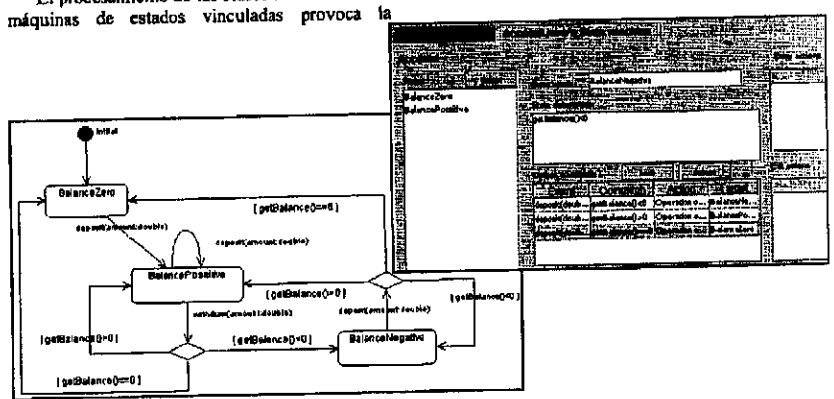


Figura 4. Posible máquina de estado para *Cuenta*, descrita en Rational Rose (superior) y nuestra herramienta(inferior)

3. Fase de ingeniería directa

La instancia del *OOS* obtenida de la etapa de ingeniería inversa, corresponde a la capa de dominio de la aplicación que será generada. Más aun, esta capa de dominio es utilizada como el punto de inicio para la generación del resto de capas.

3.1. Capa de presentación

Básicamente, la capa de presentación está compuesta por un conjunto de pantallas que permiten al usuario manipular las instancias. El formato de cada pantalla obviamente depende del tipo de sistema a desarrollar, aunque todos ellos compartan los mismos principios de diseño.

3.2. Capa de fabricaciones puras

Nuestra herramienta construye una fabricación pura [14] por cada clase de dominio. Sus métodos son los encargados de gestionar las relaciones de la clase de dominio con otras clases de dominio. Para generar las fabricaciones puras, la herramienta procesa todas las relaciones contenidas en la instancia del *OOS*, añadiendo algunos métodos a la clase principal (correspondiente a la tabla principal en la relación de clave ajena original).

3.3. Capa de persistencia

La capa de persistencia tiene sólo una clase que centraliza el acceso a la base de datos desde la capa de dominio. Esta clase juega el papel de Agente de Base de Datos (*Broker Database*) [6].

3.4. Otras capas

Dependiendo del tipo de aplicación a generar, el sistema final puede tener otras capas adicionales que incluyan otras funcionalidades. En las aplicaciones JSP estándar y basadas en EJBs, por ejemplo, cuando el usuario pulsa un botón en la página Web para ejecutar una operación, los datos son recogidos por un *servlet* que se ejecuta en el servidor; luego, el *servlet* instancia la correspondiente clase o EJB, invoca los correspondientes métodos *"set"* para establecer en los campos de las instancias los valores deseados y, entonces, ejecuta sobre la instancia el método correspondiente al botón presionado (*insert*, *update*... o *deposit*, *withdraw*, etc.).

```
public boolean statels_NEGATIVE(Broker bd){···}
public boolean statels_POSSITIVE(Broker bd){···}
public void deposit(Broker bd, double amount)
{
  if (statels_ZERO(bd)) {
    mBalance+=amount;
    setState_POSSITIVE(bd); return;
  }
  if (statels_NEGATIVE(bd) &&
getBalance(bd)+amount>0) {
    mBalance+=amount;
    setState_POSSITIVE(bd); return;
  }
  if (statels_NEGATIVE(bd) &&
getBalance(bd)+amount==0) {
    mBalance+=amount;
    setState_ZERO(bd); return;
  }
  if (statels_NEGATIVE(bd) &&
getBalance(bd)+amount<0) {
    mBalance+=amount;
    setState_NEGATIVE(bd); return;
  }
  if (statels_POSSITIVE(bd)) {
    mBalance+=amount;
    setState_POSSITIVE(bd); return;
  }
}
```

Figura 5. Campos y métodos añadidos, procedentes de la Figura 4

Adicionalmente, la herramienta añade en la aplicación final un conjunto de clases fijas con el objetivo de gestionar la seguridad, los roles, usuarios, etc. Tanto en las aplicaciones basadas en JSP estándar como en las basadas en EJB, por ejemplo, un conjunto de páginas Web y *servlets* son añadidos a la capa de presentación para permitir la creación de usuarios, asignación de permisos, etc., preservando siempre la separación entre la lógica de negocio (la lógica de seguridad, en este caso).

3.5. Metamodelos dependientes de la plataforma

El *OOS* encapsula las estructuras necesarias para representar diagramas genéricos de clases. Sin embargo, cada plataforma final tiene particularidades que aconsejan utilizar metamodelos específicos a la plataforma.

En el caso de los EJBs, por ejemplo, por cada tabla, se añade un *Entity EJB* a la capa de dominio. No obstante, para que los clientes remotos puedan encontrar y acceder al EJB, tenemos que añadir una interfaz *"home"*; de la misma manera, también es necesario agregar una interfaz *"remote"* para invocar a los métodos de negocio. En este caso, es preciso tener en cuanta la existencia de un nuevo elemento, la interfaz, que es un elemento que no habíamos considerado en la descripción del *OOS*. Del mismo modo, en el metamodelo de C# se han incluido las propiedades, que son un tipo especial de métodos que no necesitan parámetros ni tienen paréntesis.

Además del metamodelo genérico, la herramienta dispone de otras especializaciones que implementan las particularidades propias de los lenguajes de generación, como Java, que incluye el tipo *JavaClass* donde se incluye un campo *isInterface* para especificar si la clase es una interfaz o no.

4. Trabajos relacionados

En el ámbito de la reingeniería, las bases de datos han sido ampliamente estudiadas como elemento vital de las empresas, susceptible de quedar desfasado tecnológicamente, o degradado por las sucesivas tareas de mantenimiento, esto es, como sistema heredado que es.

Una de las tareas que con más frecuencia se acontece en este ámbito es la *traducción de bases de datos*, en la que podemos distinguir tres tareas principales [9]: traducción del esquema, migración de los datos, y traducción del código. Cuando se pretende migrar una base de datos antigua (colecciones de ficheros, en red, etc.) también hay que tener en cuenta que gran parte de la lógica para la gestión de estos sistemas heredados reside en el código fuente de las aplicaciones que manejan sus datos.

En [9] se describe un entorno, MIDAS, que aborda la migración de una base datos en red hacia el modelo relacional. MIDAS, además de ofrecer soporte para este cambio de base de datos, permite sustituir las subrutinas de acceso a la base de datos mediante sentencias SQL mediante el uso de grafos que representan las subrutinas y operaciones de transformación llevadas a cabo sobre los grafos.

En otros trabajos [12, 13], se trata DB-MAIN, un proyecto para la reingeniería de bases de datos. Los autores describen una metodología genérica que se compone de dos fases, (1) *extracción de la estructura de los datos* y (2) *conceptualización de las estructuras de datos*. Para ofrecer soporte a esta metodología, se desarrolló una herramienta que recibe el mismo nombre que el proyecto en el que se ubica, DB-MAIN. Esta herramienta soporta la metodología y muchas otras actividades de reingeniería sobre bases de datos, incluyendo la gestión de la propagación de los cambios realizados en la base de datos hacia el código fuente de la aplicación que soporta el acceso a los datos que almacena.

5. Conclusiones y trabajo futuro

En este artículo hemos presentado algunas características de una herramienta para la generación automática de aplicaciones multicapa a partir de bases de datos relacionales mediante un proceso completo de reingeniería. Las sucesivas transformaciones están basadas en técnicas de descripción formal en el sentido de [5], en [11] y [18] se encuentran detallados los metamodelos utilizados en el proceso y algunas operaciones del mismo.

Basada en un excelente diseño arquitectónico, la herramienta puede generar cuatro tipos diferentes de aplicaciones a partir de cuatro tipos de bases de datos. Además, su diseño facilita la adición de nuevos gestores de bases de datos, para tomar sus bases de datos como punto de partida, así como la implementación de nuevos generadores de código.

La herramienta ha sido usada en el desarrollo de, al menos 12 proyectos (Intranet de la facultad de Ciencias Químicas, parte pública de la Web de la Escuela Superior de Informática, que es dinámica y se nutre de los datos que almacenan los profesores en la base de datos, también podemos citar la Intranet del grupo de investigación Alarcos; todos en el campus de Ciudad Real de la UCLM, además de otros proyectos.). La estructura uniforme del código generado, y la adecuada ubicación de las responsabilidades en las clases permiten a los desarrolladores decrementar la curva de aprendizaje de las aplicaciones generadas, obteniendo por ello importantes ahorros en lo que al tiempo y la economía se refiere durante la etapa de mantenimiento de las aplicaciones. Uno de los proyectos esta bajo nuestro control desde hace más de dos años. Este incluye una base de datos de más de 80 tablas que son gestionadas tanto por una aplicación basada en páginas JSP estándar

como por una aplicación de escritorio estándar de Java. En este caso, ambas aplicaciones comparten las capas de dominio y persistencia, con lo que se evita realizar por duplicado las tareas de mantenimiento. El tiempo medio necesario para llevar a cabo una operación de mantenimiento correctivo y perfectivo es inferior a 1.5 horas.

Actualmente, estamos implementando un generador para obtener aplicaciones Web .NET que puedan compartir también la capa de dominio. Además, estamos trabajando en el desarrollo de múltiples componentes para almacenar todos los modelos en el formato estándar XMI, que nos permitirá su manipulación y representación mediante otras herramientas.

En nuestra opinión, la combinación de técnicas de descripción formal con la arquitectura dirigida por modelos, es una excelente combinación que mejora la automatización de los procesos software.

6. Referencias

[1] Alvaro, A., D. Lucrédio, y V. Cardoso García. *Orion-RE: A Component-Based Software Reengineering Environment*. in *10th Working Conference on Reverse Engineering (WCRE'03)*. 2003. Canada: IEEE Computer Society.

[2] Arnold, R.S., *Software Reengineering*, 0-8186-3272-0. 1992: IEEE Press.

[3] Biggerstaff, B.G. y D.E. Mitbander, *Program understanding and the concept assignment problem*. Communications of the ACM, 1994. 37(5). pp. 72-83.

[4] Brown, K. y B.G. Whitenack, *Crossing Chasms, A Pattern Language for Object-RDBMS Integration*, K.S.C. Editor. 1995.

[5] Broy, M., *Towards a Mathematical Foundation of Software Engineering Methods*. IEEE Transactions on Software Engineering, 2001. 27(1). pp. 42-57.

[6] Buschman, F., et al., *A System of Patterns: Pattern-Oriented Software Architecture*. 1996: Addison Wesley.

[7] Canfora, G., A. Cimitile, A. De Lucia, y G.A. Di Lucca. *Decomposing Legacy Programs: A First Step Towards Migrating to Client-Server Platforms*. in *6th International Workshop on Program Comprehension (IWPC'98)*. 1998. Ischia, Italy: IEEE.

[8] Chiang, R., T. Barron, y V.C. Storey, *Reverse engineering of relational databases: extracting of an EER model from a relational database*. Journal of Data and Knowledge Engineering, 1994. 12(2). pp. 107-142.

[9] Cohen, Y. y Y.A. Feldman, *Automatic High-Quality Reengineering of Database Programs by Abstraction, Transformation and Reimplementation*. ACM Transactions on Software Engineering and Methodology, 2003. 12(3). pp. 285-316.

[10] Gamma, E., R. Helm, J. Johnson, y J. Vlissides, *Design Patterns Elements of Reusable of Object-Oriented Software*. 1995: Addisson-Wesley.

[11] García-Rodríguez de Guzmán, I., M. Polo, y M. Piattini. *Metamodels and architecture of an automatic code generator*. in *Proceedings of 2nd Nordic Workshop on the Unified Modeling Language (NWUML'2004)*. 2004. Turku (Finlandia): TUCS General Publication.

[12] Henrard, J., V. Englebert, J.M. Hick, D. Roland, y J.L. Hainaut. *Program understanding in database reverse engineering*. 2002.

[13] Hick, J.-M. y J.-L. Hainaut, *Strategy for Database Application Evolution: The DB-MAIN Approach*. LNCS 2813, 2003. pp. 291-306.

[14] Larman, C., *Applying UML and Patterns*. 1998, New York: Prentice Hall, Upper Saddle River.

[15] Leavit, N., *Whatever happened to object-oriented databases?* IEEE Computer, 2001. 33(8). pp. 16-19.

[16] Martin, J. y H.A. Müller. *Strategies for Migration from C to Java*. in *Fifth European Conference on Software Maintenance and Reengineering (CSMR'01)*. 2001. Lisbon, Portugal: IEEE Computer Society.

[17] Pedro de Jesus, L. y P. Sousa. *Selection of Reverse Engineering Methods for Relational Dabases*. in *Proceedings of the Third European Conference on Software Maintenance*. 1998. Los Alamitos, California: Nesi, Verhoef.

[18] Polo, M., I.Á. Gómez, M. Piattini, y F. Ruiz, *Generating three-tier applications from relational databases: a formal and practical approach*. Information & Software Technology, 2002. 44(15). pp. 923-941.