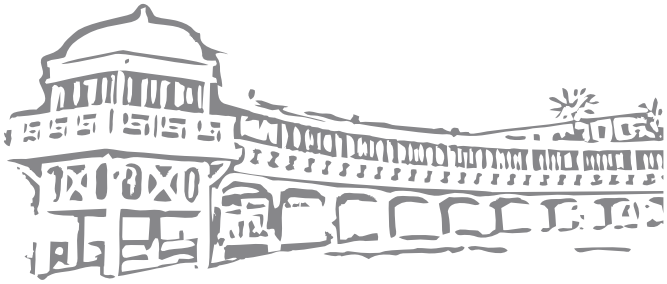


Jornadas Sistedes 2014

Cádiz, del 16 al 19 de septiembre

JISBD PROLE JCIS DC

XIX Jornadas de Ingeniería del Software y Bases de Datos

ACTAS

XIX Jornadas de Ingeniería del Software y Bases de Datos

Editores:

Javier Tuya
Mercedes Ruiz
Nuria Hurtado

Actas de las XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD)

Cádiz, 17 al 19 de septiembre 2014

Editores: Javier Tuya, Mercedes Ruiz y Nuria Hurtado

ISBN – 10: 84-697-1152-0
ISBN – 13: 978-84-697-1152-1

Prólogo

Las XIX Jornadas de Ingeniería del Software y Bases de Datos (JISBD), se celebran en Cádiz del 16 al 19 de septiembre de 2014. JISBD forma parte de las Jornadas SISTEDES, organizadas por la Sociedad de Ingeniería del Software y Tecnologías de Desarrollo de Software. Además de JISBD, las Jornadas SISTEDES agrupan a las Jornadas sobre Programación y Lenguajes (PROLE) y las Jornadas sobre Ciencia en Ingeniería de Servicios (JCIS), con las que la comunidad JISBD mantiene una estrecha relación.

JISBD se han consolidado, gracias al esfuerzo de nuestra comunidad, como un foro de intercambio y promoción de conocimiento en el área de Ingeniería del Software y Bases de Datos en el ámbito español, portugués y latinoamericano. En JISBD 2014, como en ocasiones anteriores, se han presentado cuatro tipos de contribuciones: artículos regulares, artículos de investigación emergente, artículos relevantes ya publicados y demostraciones de herramientas. Este año se cuenta con 50 contribuciones (14 regulares, 15 emergentes, 12 ya publicados y 9 demos), con un ratio de aceptación del 87% en regulares y 88% en emergentes.

Estas contribuciones se han repartido en las seis sesiones temáticas habituales de JISBD:

- 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, Proceso Software y Metodologías.
- Sesión 4: Calidad, Pruebas y Requisitos.
- Sesión 5: Desarrollo de Software Dirigido por Modelos.
- Sesión 6: Líneas de Producto, Componentes y Arquitecturas Software.

Además, este año se han incorporado dos importantes novedades:

- Un taller de emprendimiento en las TIC que cuenta con la participación de varios casos de éxito de emprendedores así como la presentación de iniciativas de apoyo y financiación.

- Un Doctoral Consortium para que los doctorandos en el ámbito de SISTEDES presenten su proyecto de tesis doctoral y reciban realimentación por parte de expertos en un comité de evaluación

Se cuenta asimismo con dos conferencias invitadas: “Software Testing and/or Software Monitoring: Differences and Commonalities” ofrecida por Antonia Bertolino, Directora de Investigación del Italian National Research Council (CNR) en Pisa y “Automatic Identification of Service Candidates from Business Process Models” ofrecida por Jan Mendling, Director de Institute for Information Business en WU Vienna.

Me gustaría destacar y agradecer especialmente la importante contribución María José Escalona y Pepe Riquelme, responsables de las nuevas actividades que se han incorporado (el taller de emprendimiento y el Doctoral Consortium) que abordaron gustosamente esta nueva

tarea con gran determinación. También destacar el papel de los miembros del Comité de Programa, cuyas revisiones dejan entrever el importante esfuerzo que han realizado para enriquecer los trabajos y promover la discusión científica. Finalmente, quiero subrayar el papel de los Coordinadores de Sesiones, en los que he delegado parte del trabajo, para hacer de éstas un foro interactivo de intercambio y generación de conocimiento, así como la labor realizada por los Comentaristas de los artículos en las distintas sesiones.

Pero JISBD 2014 no habrían dado su fruto sin el trabajo realizado por todos los miembros del Comité Organizador de la Universidad de Cádiz, presidido por Mercedes Ruiz y el apoyo de los miembros del Comité Permanente, a los que tengo que agradecer que hayan depositado en mí su confianza permitiéndome actuar como Presidente del Comité de Programa de JISBD 2014. Adicionalmente, tengo que mencionar la estrecha colaboración mantenida con los Presidentes de los Comités de Programa de los dos años anteriores, Ana Moreno y Antonio Ruiz, cuyos inestimables consejos han sido vitales.

No puedo terminar sin manifestar mi gratitud a los autores de los distintos trabajos, así como a todos los asistentes a las Jornadas, por contribuir al éxito y a la consolidación de las mismas.

Cádiz, Septiembre de 2014

Javier Tuya

Presidente del Comité de Programa de JISBD 2014

Prólogo de la organización

La décima novena edición de las Jornadas de Ingeniería del Software y Bases de Datos (JISBD) se celebra en 2014 en la ciudad de Cádiz, en el ámbito de las Jornadas Sistedes 2014. Como organizadores, agradecemos al Comité Permanente la confianza depositada en nosotros y confiamos en que nuestro trabajo les permita a todos disfrutar de unas Jornadas enriquecedoras tanto desde el punto científico como personal.

Queremos agradecer a los miembros del comité de programa de JISBD su disponibilidad y trabajo efectuado durante las revisiones y la confección del programa científico del evento, así como a los coordinadores de las diferentes sesiones su atención a los detalles particulares de cada una de ellas. Es imprescindible manifestar la excelente labor del presidente del comité de programa, Javier Tuya, que, con su implicación en todos los detalles relacionados con las Jornadas, nos ha facilitado en gran medida nuestra tarea.

También es necesario agradecer a las entidades colaboradoras su contribución en la celebración del evento. En tiempos como los actuales resulta, si cabe, aún más importante este agradecimiento. Agradecemos a la Universidad de Cádiz, a la Escuela Superior de Ingeniería y al Departamento de Ingeniería Informática las facilidades que nos han dado para el uso de las instalaciones y recursos. Queremos reconocer la importante contribución de los grupos de investigación de Mejora del Proceso Software y Métodos Formales y UCASE de Ingeniería del Software, porque han contribuido con lo más preciado que tienen que son sus miembros. De igual manera, queremos agradecer al Excmo. Ayuntamiento de Cádiz y a su Delegación de Turismo su atenta colaboración y atención en la organización de algunas de las actividades sociales del programa de las Jornadas. De igual manera, agradecemos a las empresas Renfe e Iberia su colaboración a la hora de poner a disposición de los asistentes los descuentos para el desplazamiento a Cádiz.

Nuestra gratitud a Luis Iribarne y a Juan Manuel Vara por su disponibilidad, sus valiosos consejos y experiencia compartida sobre la organización de las dos ediciones anteriores.

Finalmente, nuestro agradecimiento más particular y especial debe ir dedicado a todos aquellos que han colaborado en los trabajos de organización de las Jornadas. Agradecemos profundamente el tiempo, las energías, la implicación, el compromiso y el trabajo de todos los que han participado en la preparación y celebración de las Jornadas.

Os deseamos a todos una feliz estancia en Cádiz y confiamos en que las Jornadas constituyan un excelente foro de intercambio de conocimientos, experiencias y reflexión.

Cádiz, septiembre de 2014

Mercedes Ruiz

Presidenta del Comité Organizador
de las Jornadas Sistedes 2014

Comité Ejecutivo

PRESIDENTE DEL COMITÉ DE PROGRAMA
Javier Tuya (Universidad de Oviedo)

COORDINADORES DE SESIONES TEMÁTICAS
Trabajos regulares, emergentes y ya publicados

Sesión 1. Bases de Datos, Almacenes de Datos, Minería de Datos, Recuperación de la Información
Sergio Ilarri (Universidad de Zaragoza)
José Francisco Aldana (Universidad de Málaga)

Sesión 2. Ingeniería Web, Interfaces de Usuario, Sistemas Colaborativos, Computación Ubicua
Elena Navarro (Universidad de Castilla-La Mancha)
Roberto Rodríguez Echeverría (Universidad de Extremadura)

Sesión 3. Apoyo a la Decisión en Ingeniería del Software, Proceso Software y Metodologías
Mercedes Ruiz (Universidad de Cádiz)
Agustín Yagüe (Univ. Politécnica de Madrid)

Sesión 4. Calidad, Pruebas y Requisitos
Carme Quer (Universidad Politécnica de Cataluña)
Claudio de la Riva (Universidad de Oviedo)

Sesión 5. Desarrollo de Software Dirigido por Modelos
Orlando Ávila (Open Canarias)
José Raúl Romero (Universidad de Córdoba)

Sesión 6. Líneas de Producto, Componentes y Arquitecturas Software
Juan Manuel Murillo Rodríguez (Universidad de Extremadura)
David Benavides (Universidad de Sevilla)

Demos
Macario Polo (Universidad de Castilla-La Mancha)
José Antonio Cruz Lemus (Universidad de Castilla-La Mancha)

Emprendimiento en las TIC
María José Escalona (Universidad de Sevilla)

Comité de Programa

Silvia Abrahão (Univ. Polit. Valencia)
Jesús Aguilar (Univ. Pablo Olavide)
Pedro Pablo Alarcón (Univ. Polit. Madrid)
Maria José Aramburu (Univ. Jaume I)
Joao Araujo (Univ. Nova Lisboa)
José Luis Arjona (Univ. Huelva)
David Benavides (Univ. Sevilla)
Rafael Berlanga (Univ. Jaume I)
Raquel Blanco (Univ. Oviedo)
Verónica Bollati (Univ. Rey Juan Carlos)
Artur Boronat (Univ. Leicester)
Nieves Brisaboa (Univ. A Coruña)
Jordi Cabot (École des Mines de Nantes)
Paloma Cáceres (Univ. Rey Juan Carlos)
Cristina Cachero (Univ. Alicante)
Javier Cámara (Carnegie Mellon Univ.)
Carlos Canal (Univ. Málaga)
José Hilario Canós (Univ. Polit. Valencia)
Rafael Capilla (Univ. Rey Juan Carlos)
Pepe Carsí (Univ. Polit. Valencia)
José María Cavero (Univ. Rey Juan Carlos)
Pedro J. Clemente (Univ. Extremadura)
Rafael Corchuelo (Univ. Sevilla)
Dolors Costal (Univ. Polit. Catalunya)
Yania Crespo (Univ. Valladolid)
Carlos Cuesta (Univ. Rey Juan Carlos)
Valeria de Castro (Univ. Rey Juan Carlos)
Pablo de la Fuente (Univ. Valladolid)
Óscar Díaz (Univ. País Vasco)
Óscar Dieste (Univ. Polit. Madrid)
Javier Dolado (Univ. País Vasco)
M. José Escalona (Univ. Sevilla)
Joao Falcao e Cunha (Univ. Porto)
Carles Farré (Univ. Polit. Catalunya)
Manuel Fernández Bertoa (Univ. Málaga)
José Luis Fernández-Aleman (Univ. Murcia)
Eduardo Fernández-Medina (Univ. Castilla-La Mancha)
Juan Garbajosa (Univ. Polit. Madrid)
Félix García (Univ. Castilla-La Mancha)
Jesús García Molina (Univ. Murcia)
Irene Garrigós (Univ. Alicante)
Marcela Genero (Univ. Castilla-La Mancha)
Pascual González (Univ. Castilla-La Mancha)
Alfredo Goñi (Univ. País Vasco)

Miguel Goulao (Univ. Nova de Lisboa)
Esther Guerra (Univ. Autónoma de Madrid)
Francisco Gutiérrez-Vela (Univ. Granada)
Juan Hernández (Univ. Extremadura)
Arantza Illarramendi (Univ. País Vasco)
Emilio Insfrán (Univ. Polit. Valencia)
Arantza Irastorza (Univ. País Vasco)
Luis Iribarne (Univ. Almeria)
Jon Iturrioz (Univ. País Vasco)
Javier Jaén (Univ. Polit. Valencia)
Natalia Juristo (Univ. Polit. Madrid)
Marcos López (Univ. Rey Juan Carlos)
Lidia López (Univ. Polit. Catalunya)
Miguel R. Luaces (Univ. A Coruña)
María Esperanza Manso (Univ. Valladolid)
José Norberto Mazón (Univ. Alicante)
Santiago Meliá (Univ. Alicante)
Ana Moreno (Univ. Polit. Madrid)
Juan José Moreno (Univ. Polit. Madrid)
Manuel Núñez (Univ. Complutense de Madrid)
Patricia Paderewski (Univ. Granada)
Ignacio Panach (Univ. Valencia)
Óscar Pastor (Univ. Polit. Valencia)
Óscar Pedreira (Univ. A Coruña)
Vicente Pelechano (Univ. Polit. Valencia)
Jenifer Pérez (Univ. Polit. Madrid)
Mario Piattini (Univ. Castilla-La Mancha)
Macario Polo (Univ. Castilla-La Mancha)
Antonio Polo (Univ. Extremadura)
Juan Carlos Preciado (Univ. Extremadura)
Isidro Ramos (Univ. Polit. Valencia)
José Riquelme (Univ. Sevilla)
Antonio Rito (Univ. Tec. Lisboa)
José Luis Roda (Univ. La Laguna)
Daniel Rodríguez (Univ. Alcalá)
José Raül Romero (Univ. Córdoba)
Francisco Ruiz (Univ. Castilla-La Mancha)
Antonio Ruiz (Univ. Sevilla)
Roberto Ruiz (Universidad Pablo Olavide)
Goiuria Sagardui (Univ. Mondragón)
José Samos (Univ. Granada)
Fernando Sánchez (Univ. Extremadura)
Pedro Sánchez (Univ. Polit. Cartagena)
Juan Sánchez (Univ. Polit. Valencia)
Maribel Sánchez-Segura (Univ. Carlos III)
Ismael Sanz (Univ. Jaume I)
Sergio Segura (Univ. Sevilla)

Almudena Sierra-Alonso (Univ. Rey Juan Carlos)
María José Suárez-Cabal (Univ. Oviedo)
Ernest Teniente (Univ. Polit. Catalunya)
Miguel Toro (Univ. Sevilla)
Ambrosio Toval (Univ. Murcia)
Salvador Trujillo (IKERLAN)
Juan Carlos Trujillo (Univ. Alicante)
Antonio Vallecillo (Univ. Málaga)
Juan Manuel Vara (Univ. Rey Juan Carlos)
Sira Vegas (Univ. Polit. Madrid)
Belén Vela (Univ. Rey Juan Carlos)
Cristina Vicente-Chicote (Univ. Extremadura)

Comité de Organización

PRESIDENTA DEL COMITÉ ORGANIZADOR

Mercedes Ruiz Carreira

MIEMBROS DEL COMITÉ ORGANIZADOR

Juan Boubeta Puig
Mª del Carmen de Castro Cabrera
Pedro Delgado Pérez
Juan Manuel Dodero Beardo
Antonio García Domínguez
Mª Teresa García Horcajadas
Lorena Gutiérrez Madroñal
Nuria Hurtado Rodríguez
José Luis Isla Montes
Inmaculada Medina Bulo
José Miguel Mota Macías
Manuel Palomo Duarte
Elena Orta Cuevas
Guadalupe Ortiz Bellot
Carlos Rioja del Río
Iván Ruiz Rube
Alberto Gabriel Salguero Hidalgo

Entidades Colaboradoras

Índice de contenidos

Keynotes

Software Testing and/or Software Monitoring: Differences and Commonalities. *Antonia Bertolino*..... 19
Automatic Identification of Service Candidates from Business Process Models. *Henrik Leopold and Jan Mendling*..... 20

Sesión 1. Bases de Datos, Almacenes de Datos, Minería de Datos, Recuperación de la Información

Trabajos de investigación emergentes

Actualización incremental de grafos RDF a partir de bases de datos relacionales. *Liudmila Reyes-Álvarez, Yusniel Hidalgo-Delgado, Katerín Martínez-Rojas, María Del Mar Roldan-García and José F. Aldana-Montes*..... 21
Geração Automática de Esqueletos para Sistemas ETL. *Miguel Guimarães and Orlando Belo* 27
Modelado multidimensional para la visualización integrada de Big Data en plataformas de Inteligencia de Negocio. *Roberto Tardío, Elisa de Gregorio, Alejandro Maté, Rafael Muñoz-Terol, David Gil, Héctor Llorens and Juan Trujillo*..... 33

Trabajos relevantes ya publicados

Bioqueries: a Social Community for SPARQLqueries in Life Sciences. *María Jesús García Godoy, Esteban López-Camacho, Ismael Navas-Delgado and José F. Aldana Montes* 39
Adding Semantic Modules to improve Goal-Oriented Analysis of Data Warehouses using I-star. *Alejandro Maté, Juan Trujillo and Xavier Franch*..... 41

Sesión 2. Ingeniería Web, Interfaces de Usuario, Sistemas Colaborativos, Computación Ubicua

Trabajos regulares

Modelado de calidad de contexto con MLContext. *José R. Hoyos, Jesús García-Molina and Juan A. Botía*..... 43
Una solución MDE para crear aplicaciones basadas en mensajes instantáneos a través de Twitter. *Ángel Mora Segura, Juan de Lara and Jesús Sánchez Cuadrado*..... 57
Simulating Mobile Agents in Vehicular Networks. *Óscar Urra, Sergio Ilarri and Eduardo López*..... 71

Trabajos de investigación emergentes

Context-Aware Recommendations in Mobile Environments. *María del Carmen Rodríguez Hernández and Sergio Ilarri*..... 85
Herramienta Colaborativa Multidispositivo para la Edición de Modelos basada en EMF. *Miguel A. Teruel, Arturo C. Rodríguez, Elena Navarro and Pascual González*..... 91

Trabajos relevantes ya publicados

A CSCW Requirements Engineering CASE Tool: Development and Usability Evaluation. *Miguel A. Teruel, Elena Navarro, Víctor López-Jaquero, Francisco Montero and Pascual González*..... 97

Demostraciones de herramientas

WebMakeup: An End-user Tool for Web Page Customization. *Oscar Díaz, Cristóbal Arellano, Iñigo Aldalur, Haritz Medina and Sergio Firmenich*..... 99

Sesión 3. Apoyo a la Decisión en Ingeniería del Software, Proceso Software y Metodologías

Trabajos regulares

Metodología para Diseñar, Desarrollar y Evaluar una Plataforma de Entrenamiento en Desarrollo Global de Software. *Miguel J. Monasor, Aurora Vizcaino, Mario Piattini, John Noll and Sarah Beecham*..... 103
Sim-XPerience: Simulación Basada en Agentes Aplicada al Desarrollo de Software con XP. *Cristina Capitas, Nuria Hurtado and Mercedes Ruiz*..... 117
Habilidades emocionales en el sector TIC: un análisis a partir de los empleos ofertados en el sector. *Víctor Serrano, Juan M. Vara and Esperanza Marcos*..... 131

Trabajos de investigación emergentes

Una aproximación Ágil a los niveles de madurez 2 y 3 de CMMI-DEV en entornos de desarrollo Web. *C. J. Torrecilla-Salinas, J. Sedeño, M.J. Escalona and M. Mejías*..... 145

Trabajos relevantes ya publicados

Usability through Software Design. *Laura Carvajal, Ana M. Moreno, María-Isabel Sánchez-Segura and Ahmed Seffah*..... 151
A fine-grained analysis of the support provided by UML class diagrams and ER diagrams during data model maintenance. *Gabriele Bavota, Carmine Gravino, Rocco Oliveto, Andrea De Lucia, Genoveffa Tortora, Marcela Genero and José A. Cruz-Lemus*..... 153

Demostraciones de herramientas

EXEMPLAR: An Experimental Information Repository for Software Engineering Research. *José Antonio Parejo, Sergio Segura, Pablo Fernández and Antonio Ruiz Cortés*.. 155
Suite de aplicaciones colaborativas para dar soporte a la gamificación del prototipado de procesos. *Manuel Trinidad and Mercedes Ruiz*..... 159

Sesión 4. Calidad, Pruebas y Requisitos

Trabajos regulares

Entorno para la Evaluación y Certificación de la Calidad del Producto Software. *Moisés Rodríguez and Mario Piattini.*…… 163
Generación de Mutantes Válidos en el Lenguaje de Programación C++. *Pedro Delgado-Pérez, Inmaculada Medina-Bulo and Juan José Domínguez-Jiménez.*…… 177
Pruebas funcionales en programas MapReduce basadas en comportamientos no esperados. *Jesús Morán, Claudio de la Riva and Javier Tuya.*…… 191

Trabajos de investigación emergentes

Towards Multi-Objective Test Case generation for Variability-Intensive Systems. *Ana B. Sánchez, Sergio Segura and Antonio Ruiz-Cortés.*…… 205
Generación automática de casos de prueba mediante siembra automática para WS-BPEL 2.0 *Valentín Liñeiro Barea, Antonia Estero Botaro, Antonio García Domínguez and Inmaculada Medina Bulo.*…… 211

Trabajos relevantes ya publicados

Multi-dimensional Criteria for Testing Web Services Transactions. *Rubén Casado, Muhammad Younas and Javier Tuya.*…… 217

Demostraciones de herramientas

PTAC: Una herramienta para testing pasivo de sistemas con comunicaciones asíncronas. *Mª Azahara Camacho Magriñán, Inmaculada Medina Bulo and Mercedes G. Merayo.*…… 219
Demostración de NDT-Driver: una herramienta de soporte a los mecanismos de transformación de NDT. *J.A. Alberto García-García and M.J. Escalona.*…… 223

Sesión 5. Desarrollo de Software Dirigido por Modelos

Trabajos regulares

DB-Main/Models: Un caso de estudio sobre la interoperabilidad de herramientas basada en MDE. *Francisco Javier Bermúdez Ruiz, Jesús García Molina and Oscar Díaz García.*…… 227
Automatizando el desarrollo de editores gráficos cognitivamente eficaces. *David Granada, Ángel Moreno, Juan M. Vara, Verónica A. Bollati and Esperanza Marcos.*…… 241

Trabajos de investigación emergentes

SEPL: Social Environment Programming Language. Luis-María García-Rodríguez, Álvaro Gutiérrez-Pérez, Rober Morales-Chaparro, Fernando Sánchez-Figueroa……. 255
Proceso de verificación de reglas de transformación basado en métricas. *Fernando Macías, Roberto Rodríguez-Echeverría, Víctor M. Pavón, José M. Conejero and Fernando Sánchez-Figueroa.*…… 261
Primitive Operators for the Concurrent Execution of Model Transformations Based on LinTra. *Loli Burgueño, Eugene Syriani, Manuel Wimmer, Jeff Gray and Antonio Vallecillo.*…… 267

Propuesta de modelado de requerimientos en paradigmas de Ingeniería Web Ágil guiada por modelos. *J. Sedeño, C.J. Torrecilla-Salinas, M.J. Escalona and M. Mejías.*..... 273
Introducing Approximate Model Transformations. *Javier Troya and Antonio Vallecillo*.. 279
DSL-2-Browser: Un ejemplo de ejecución de un lenguaje específico del dominio en un navegador web. *Álvaro Gutiérrez-Pérez, Luis-María García-Rodríguez, Rober Morales-Chaparro and Fernando Sánchez-Figueroa.*..... 285

Trabajos relevantes ya publicados

MDD vs. traditional software development: A practitioner's subjective perspective. *Yulkeidi Martínez, Cristina Cachero and Santiago Meliá.*..... 291
PRISMA: Model-Driven Development of Aspect-Oriented Software Architectures. *Jennifer Pérez, Isidro Ramos, José A. Carsí and Cristóbal Costa-Soria.*..... 295
A Language for End-user Web Augmentation: Caring for Producers and Consumers Alike. *Oscar Díaz, Cristóbal Arellano and Maider Azanza.*..... 297

Demostraciones de herramientas

Herramienta de soporte en procesos de modernización, para las fases de ingeniería inversa y reestructuración. *Víctor M. Pavón, Roberto Rodríguez-Echeverría, Fernando Macías, Pedro J. Clemente and Fernando Sánchez-Figueroa.*..... 299
CERVANTES: Un framework para el diseño y desarrollo de sistemas distribuidos. *M.A. Barcelona, L. García-Borgoñón, J.I. Calvo and M.J. Escalona.*..... 303

Sesión 6. Líneas de Producto, Componentes y Arquitecturas software

Trabajos regulares

Análisis de la aplicabilidad de medidas software para el diseño semi-automático de arquitecturas. *Aurora Ramírez, José Raúl Romero and Sebastián Ventura.*..... 307
Propuesta de metodología de despliegue de aplicaciones en nubes heterogéneas con TOSCA. *José Carrasco, Javier Cubo and Ernesto Pimentel.*..... 321
Configurable feature models. *Pablo Trinidad, Antonio Ruiz-Cortés and Jesús García-Galán.*..... 335

Trabajos de investigación emergentes

Un Enfoque Basado en Modelos para incorporar Requisitos No Funcionales y de Integración de Software en el Diseño de Arquitecturas Orientadas a Servicios. *M. Guessous, M.A. Barcelona, L. García-Borgoñón and M. Alba.*..... 349

Trabajos relevantes ya publicados

People as a Service: a mobile-centric model for providing collective sociological profiles. *José García-Alonso, Javier Miranda, Javier Berrocal, Juan Manuel Murillo and Carlos Canal.*..... 355
Change-Impact driven Agile Architecting. *Jessica Díaz, Jennifer Pérez, Juan Garbajosa and Agustín Yagüe.*..... 357
Self-Adaptation of Mobile Systems with Dynamic Software Product Lines. *Nadia Gámez, Lidia Fuentes and José María Troya.*..... 359

Demostraciones de herramientas

Automated Analysis of Diverse Variability Models with Tool Support. *Fabricia Roos-Frantz, José A. Galindo Duarte, David Benavides, Antonio Ruiz Cortés and Jesús García-Galán.…….....* 361
WindRose: A Cloud Based IDE for the Automated Analysis of Feature Models. *José A. Galindo, David Benavides, Mauricio Alférez, Mathieu Acher and Benoit Baudry.…….....…* 365

Software Testing and/or Software Monitoring: Differences and Commonalities

Antonia Bertolino

ISTI-CNR
Via Moruzzi, 1
Pisa, Italy
antonia.bertolino@isti.cnr.it

Abstract. Validation is an essential part of the software life cycle. The actual functional and non-functional behaviour of an application needs to be checked against the expected or intended behaviour. Both testing and monitoring are widely used approaches for this purpose. More traditionally testing is considered as a technique for fault removal and forecasting during development. Monitoring is instead conceived for run-time observation of deployed software. Testing and monitoring approaches are usually contrasted as being, respectively, in-the-laboratory vs. in-the-field, and active vs. passive. In this talk I will overview concepts and techniques for software testing and monitoring, and will discuss how for modern pervasive and dynamic software systems the two approaches tend to converge in combined and synergic ways.

Automatic Identification of Service Candidates from Business Process Models

Jan Mendling

Wirtschaftsuniversität Wien
Welthandelsplatz 1
1020 Vienna, Austria
jan.mendling@wu.ac.at

Abstract. Business process models play an important role in the design and analysis of business operations. For this reason, many companies develop and maintain repositories in which models of business processes are centrally stored. In this talk, I will present analysis techniques for automatically extracting knowledge from such process repositories. The focus will be on leveraging these models for automatically identifying service candidates. Results are presented of applying these techniques on various process repositories from practice.

Actualización incremental de grafos RDF a partir de bases de datos relacionales

Liudmila Reyes-Álvarez[1], Yusniel Hidalgo-Delgado[1], Katerín Martínez-Rojas[1], María del Mar Roldán-García[2], José F. Aldana-Montes[2]

[1]Universidad de las Ciencias Informáticas, Carretera a San Antonio de los Baños, Km. 2 1/2. Torrens, La Lisa. La Habana, Cuba
[2]Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga, España.
{lreyes, Yhdelgado}@uci.cu; kmrojas@estudiantes.uci.cu; {mmar, jfam}@lcc.uma.es

Resumen. El modelo de datos relacional constituye el paradigma utilizado con mayor frecuencia por los sistemas operacionales que actualmente utilizan las empresas para gestionar sus procesos. Por otra parte, las tecnologías de la web semántica han logrado un importante impulso en los últimos años. Se han desarrollado estándares tales como *Resource Description Framework* (RDF) y *Web Ontology Language* (OWL). RDF constituye el modelo de datos utilizado para la publicación y enlazado de datos estructurados en la Web, un enfoque denominado datos vinculados (*Linked Data*) y que constituye la base sobre la que se sustenta la Web Semántica. Estudios recientes demuestran que es posible trasladar el modelo de datos relacional al modelo de datos basado en grafos RDF. Por tal motivo es posible realizar operaciones de inserción, actualización y eliminación en ambos sentidos relacional-grafo y grafo-relacional. Aproximaciones existentes solo se suscriben a la actualización total de los grafos RDF, es decir, un cambio generado en la base de datos relacional implica que sea necesario generar nuevamente el grafo RDF en su totalidad, incurriendo en altos costos de tiempo y recursos computaciones. Es por tanto necesario la definición de nuevos métodos que detecten estas actualizaciones y luego las trasladen al modelo de datos RDF. En este artículo se propone un modelo para la actualización incremental de grafos RDF a partir de la detección de cambios en las bases de datos relacionales, evitando así la generación total del grafo RDF.

Palabras clave: actualización incremental, modelo relacional, modelo RDF, Web Semántica.

1 Introducción

Las tecnologías de la Web Semántica han permitido crear almacenes de datos en la Web, construir vocabularios, y escribir las reglas para el manejo de datos. La base sobre la que se sustenta la Web Semántica son los principios de datos vinculados (Linked Data) [1]. Los datos vinculados utilizan varios estándares propuestos por la *World Wide Web Consortium* (W3C), tales como: *Resource Description Framework*

(RDF), SPARQL (acrónimo recursivo del inglés *SPARQL Protocol and RDF Query Language*) y *Web Ontology Language* (OWL).

La mayoría de los datos en los que se basa la Web actual se almacenan en bases de datos relacionales debido a su probada trayectoria. Los datos representados en RDF pueden ser interpretados, procesados y razonados por los agentes de software. De ahí que el cambio del modelo relacional al modelo RDF para la manipulación de la información sea uno de los retos de investigación actuales.

Los grupos de investigación en Web Semántica de las Universidad de las Ciencias Informáticas de Cuba y de la Universidad de Málaga están desarrollando en colaboración el proyecto DBJournal. Este proyecto tiene como objetivo la publicación de metadatos bibliográficos siguiendo los principios de los datos vinculados. En el contexto del este proyecto se ha desarrollado la plataforma BM2LOD [3]. Esta plataforma integra fuentes de datos de revistas académicas que emplean el protocolo de intercambio de metadatos *Open Archive Initiative Protocol for Metadata Harvesting (OAI-PMH[1])*. Dichos metadatos se almacenan en una base de datos relacional usando la herramienta Metharto [4] y se alinean con el modelo RDF generando las tripletas correspondientes. La carencia fundamental de la plataforma BM2LOD se basa en la actualización del grafo RDF, generándolo en su totalidad una vez que se detectan cambios en la base de datos relacional [3]. De ahí que el objetivo de este artículo sea proponer un nuevo método para la actualización incremental de grafos RDF generados a partir de una base de datos relacional. Este artículo se organiza como sigue: la sección 2 presenta algunos trabajos que proponen la generación de RDF a partir de bases de datos relacionales. La sección 3 describe el método de actualización incremental que se propone y las principales innovaciones que se aprecian en este nuevo enfoque. Por último, la sección 4 discute las conclusiones y las líneas de trabajo futuro.

2 Trabajos Relacionados

Existen variados trabajos que demuestran la posibilidad de trasladar el modelo de datos relacional al modelo de datos basado en grafo RDF. El W3C propone lenguajes para establecer una alineación entre ambos modelos. Específicamente en [2] se realiza una comparación entre los lenguajes de alineación existentes, clasificándolos de acuerdo a su propósito y características, y se ofrece una guía para elegir un lenguaje de alineación según su objetivo. En [5] se propone un mecanismo de traslación bidireccional Relacional–RDF y se describe la herramienta D2RQ++, la cual es una extensión de D2RQ. Anteriormente D2RQ generaba solamente las tripletas RDF y las mostraba. Actualmente, mediante D2RQ++, es posible actualizar los datos de la base de datos relacional a partir de los cambios que existan en las tripletas RDF y viceversa. Sin embargo, este proceso de actualización no es incremental y además el lenguaje de alineación que usa la herramienta no es un estándar del W3C. En nuestra propuesta se contempla el uso de R2RML, el lenguajes de alineación propuesto por el W3C [8].

[1] http://www.openarchives.org/pmh/

Otro trabajo a tener en cuenta es la herramienta Morph-RDB [7], que es un motor RDB2RDF desarrollado por el Ontology Engin*eering Group*[2]. Morph-RDB genera un *dataset* RDF virtual a partir de una base de datos relacional de acuerdo con una alineación definida entre los dos modelos que se especifica con el lenguaje R2RML. Entre los trabajos más relevantes se encuentra *Virtuoso RDF Views* [6], que realiza la alineación del modelo relacional con el modelo RDF, y ofrece la posibilidad de personalizar dicho modelo RDF. *Virtuoso* incluye un metalenguaje declarativo propio para definir la alineación de datos en bases de datos relacionales con ontologías RDF. La alineación es dinámica, lo que implica que los cambios en los datos subyacentes se reflejan inmediatamente en la representación RDF [6]. La diferencia fundamental entre estas dos últimas propuestas y nuestro método es que las primeras trabajan con vistas RDF de la base de datos relacional completa que se crean dentro de la propia herramienta. Nuestro método, sin embargo, genera nuevas tripletas RDF basándose en una vista de la base de datos relacional que se genera a partir de los cambios detectados en la misma, utilizando para ello un lenguaje de alineación estándar.

3 Método de Actualización Incremental

El objetivo fundamental de nuestra propuesta es definir un método para actualizar un grafo RDF, construido a partir de una base de datos relacional, cuando se produce un cambio en dicha base de datos, con el menor coste en tiempo y recursos computacionales posible. De ahí que sea relevante seguir un enfoque incremental, es decir evitar la repetición de procesos desde cero y aprovechar los resultados existentes antes del cambio.

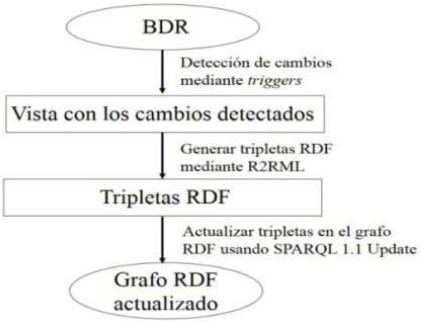


Fig. 1. Procedimiento para la actualización incremental de grafos RDF

La Figura 1 muestra una descripción del método propuesto para la actualización incremental de grafos RDF. Las fases que conforman la propuesta son: (1) Detectar cambios en la base de datos relacional. (2) Generar las tripletas RDF que representan dichos cambios. (3) Actualizar el grafo RDF con las tripletas generadas. A continuación se describe cada una de las fases que conforman la propuesta de solución.

[2] http://www.oeg-upm.net/

Detectar cambios en la BDR

Para detectar los cambios en la base de datos relacional se usan *triggers* que capturan los datos involucrados en las operaciones de inserción, eliminación y/o modificación realizadas sobre las tablas. Para almacenar los datos de dichos cambios se crea una tabla auxiliar, cuyos atributos hacen referencia al nombre de la tabla modificada, el identificador de la fila donde se realizó el cambio, así como el tipo de modificación realizado (inserción, eliminación o modificación).

A cada tabla del modelo relacional se asocian dos *triggers*, uno para las operaciones de inserción y modificación y otro para la eliminación. La diferencia entre ambos radica en capturar el identificador nuevo de la fila creada o el identificador de la fila eliminada, que ya no existiría. Ambos *triggers* han de invocar la misma función que se encarga de insertar los datos descritos anteriormente en la tabla auxiliar. Cada vez que los cambios recogidos en la tabla auxiliar sean alineados con el modelo de datos RDF se borrarán las tuplas de la tabla, evitando el cúmulo de información innecesaria en la base de datos relacional.

Generar tripletas RDF

Para generar las tripletas RDF se hace uso del lenguaje de alineación R2RML, que es el lenguaje de alineación estándar del W3C [2]. Las ventajas que se tuvieron en cuenta para decidir utilizar R2RML como lenguaje de alineación en nuestra propuesta son: 1) Utiliza ontologías de dominio para modelar los datos existentes en la base de datos relacional. 2) Se basa en el concepto de tablas lógicas, que pueden ser una tabla, una vista o consulta SQL válida. 3) Soporta transformación de datos, cálculo de datos o filtros antes de generar las tripletas. 4) Permite a los usuarios definir manualmente las asignaciones. 5) El proceso se considera semiautomático ya que se puede modificar la alineación, adquiriendo ésta un nivel de personalización.

Para realizar la alineación mediante R2RML es necesario generar un documento de alineación que hace referencia a la estructura de las tablas de la base de datos relacional para obtener los datos contenidos en ellas [8].

En esta fase de la propuesta, la primera vez que se realiza la alineación, los datos se extraen directamente de las tablas del modelo relacional y se genera el grafo RDF correspondiente. Cuando se proceda a realizar la actualización de la base de datos relacional, las tripletas serán generadas a partir de la tabla auxiliar generada en la fase anterior.

Actualizar tripletas en el Grafo RDF

De la fase anterior se obtienen las tripletas RDF y se conoce a priori el tipo de operación que implicó el cambio en la base de datos relacional. Por lo tanto, es posible realizar las consultas SPARQL de inserción, eliminación o modificación correspondientes, que se encargarán de reflejar los cambios identificados en la base de datos relacional en el grafo RDF. Con la nueva versión SPARQL 1.1 *Update* es posible realizar ese tipo de consultas. Esta es la última fase que arroja como resultado el grafo RDF actualizado. Este grafo no se genera completamente, sino que se actualiza de forma incremental.

A continuación se presenta un ejemplo ilustrativo de las distintas fases del método propuesto. Dada la tabla *Author*, se obtienen las correspondientes tripletas RDF mediante la alineación con el modelo ontológico correspondiente (ver Figura 2), usando la implementación del lenguaje de alineación R2RML (*R2RML-Parser)*.

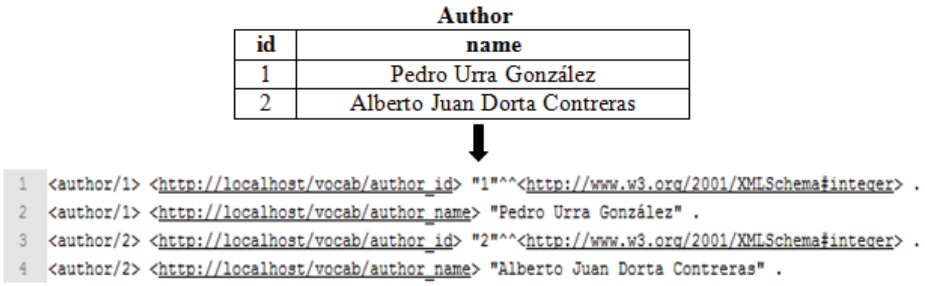


Fig. 2. Tabla *Author* de la base de datos relacional y sus tripletas RDF correspondientes

Cuando se realiza alguna operación de inserción, modificación o eliminación en la tabla *Author* (ver Figura 3) se ejecuta un *trigger* que invoca una función que inserta los datos referentes a los cambios en una tabla *check_updates*. Mediante una consulta *select* a la tabla *check_updates* se obtiene la tupla correspondiente al cambio detectado y mediante *R2RML-Parser* se obtienen las tripletas RDF. Posteriormente, a partir de las tripletas generadas anteriormente y del tipo de operación almacenada en la tabla *check_updates*, se crean las consultas SPARQL dinámicas.

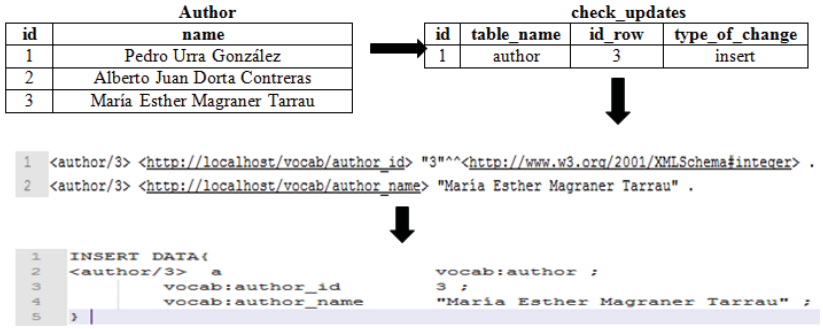


Fig. 3. Ejemplo que ilustra la actualización de la tabla *Author* de la base de datos relacional.

4 Conclusiones y trabajos futuros

En este artículo se describe un método para la actualización incremental de grafos RDF, generados a partir de una base de datos relacional, cuando se producen cambios en la misma. El método propuesto almacena los datos de los cambios en una tabla auxiliar mediante el uso de *triggers*, lo cual mejora el rendimiento de la base de datos. El modelo propuesto evita la generación total del grafo RDF, por lo que aporta una

solución viable a los problemas de escalabilidad y rendimiento en la modificación y consulta de los datos procedentes de una base de datos relacional. Actualmente el método cuenta con una implementación parcial en la plataforma BM2LOD donde se enfrenta a los siguientes desafíos:1) Luego de una eliminación asegurar que no queden triples inconsistentes. 2) La gestión eficientemente de los borrados en cascada y, 3) luego de una modificación de las triples asegurar que no se ha perdido ningún dato y que el resultado que se obtiene es equivalente al que se alcanzaría al generar un nuevo grafo RDF a partir del estado actual de la base de datos.

Como línea de trabajo futuro principal a seguir se tiene la profundización en las características propias de cada uno de los cambios considerados y su adaptación al contexto donde ocurra el cambio. Una vez se haya concluido esta labor el objetivo final será la implementación del método propuesto en la plataforma BM2LOD y la realización de pruebas de evaluación entre ésta y otras plataformas seleccionadas para el análisis.

Agradecimientos

Parcialmente financiado por los proyectos TIN2011-25840 (Ministerio de Ciencia e Innovación) y P11-TIC-7529 y P12-TIC-1519 (Plan Andaluz de Investigación, Desarrollo e Innovación).

Referencias

1. Heath, T., Bizer, C.: Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology 1 (2011) 1-136
2. Hert, M., Reif, G., Gall, H.C.: A comparison of rdb-to-rdf mapping languages. In: Proceedings of the 7th International Conference on Semantic Systems, ACM (2011) 25-32
3. Hidalgo-Delgado, Y., Reyes-Álvarez, L., Leiva-Mederos, A., Roldán-García, M.d.M., Aldana-Montes, J.F.: Bm2lod: Platform for publishing bibliographic data as linked open data. In Philip Powell, M.B.N., Isaas, P., eds.: Proceedings of 7th IADIS International Conference on Information Systems 2014. 978-989-8704-04-7, IADIS Press (2014)
4. Hidalgo-Delgado, Y., Rodríguez Puente, R., Ortiz Muñoz, E., Alonso Sierra, L.E.: Herramienta para la recolección de metadatos bibliográficos mediante el protocolo oai-pmh. In: Proceedings of the II Conferencia Internacional de Ciencias Computacionales de Informáticas, XV Convención y Feria Internacional Informática 2013 (2013)
5. Ramanujam, S., Khadilkar, V., Khan, L., Seida, S., Kantarcioglu, M., Thuraisingham, B.: Bi-directional translation of relational data into virtual rdf stores. In: Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on, IEEE (2010) 268-276
6. Erling, O., Mikhailov, I.: Virtuoso: RDF support in a native RDBMS. Springer (2010)
7. Priyatna, F., Corcho, O., Sequeda, J.: Formalisation and experiences of r2rml-based sparql to sql query translation using morph. In: Proceedings of the 23rd international conference on World wide web, International World Wide Web Conferences Steering Committee (2014) 479-490
8. Das, S., Sundara, S., Cyganiak, R.: R2rml: Rdb to rdf mapping language. w3c recommendation, 27 september 2012 (2013)

Geração Automática de Esqueletos para Sistemas ETL

Miguel Guimarães e Orlando Belo

Centro de I&D ALGORITMI
Departamento de Informática, Escola de Engenharia, Universidade do Minho
Campus de Gualtar, 4710-057 Braga, PORTUGAL

Resumo. O projeto de um sistema de povoamento obedece a regras de concepção e implementação bastante específicas, bem como integra um variadíssimo leque de regras de trabalho abrangendo um grande número de tarefas de extração, transformação e carregamento de dados. Com base em experiências práticas de aplicação, hoje sabemos que uma abordagem orientada por padrões de ETL facilita o seu projeto, desenvolvimento e exploração, além de facilitar muito a sua compreensão. Neste trabalho, abordamos esse tipo de aproximação e apresentamos uma forma de produzir automaticamente "esqueletos" para sistemas de ETL, a partir da sua especificação em redes de Petri coloridas. Além de expormos a base de definição e construção de padrões de ETL, bem como a sua especificação em redes de Petri coloridas, apresentamos e discutimos também a forma como podemos providenciar de forma automática a implementação física dos referidos esqueletos.

Palavras-Chave: *Data Warehousing Systems*, Modelação Conceptual, Lógica e Física de Sistemas de ETL, Redes de Petri Coloridas e Kettle.

1 Introdução

Há muito que os profissionais de *data warehousing* reconhecem a importância e a criticidade da implementação de um sistema de povoamento de um *data warehouse*, vulgarmente reconhecido como sistema de ETL (*Extract-Transform-Load*), no ciclo de desenvolvimento de um *sistema de data warehousing* (SDW). O sistema de ETL é apenas um dos componentes do sistema global, mas mesmo assim é o responsável pelo consumo de cerca de 70% de todos os recursos envolvidos na implementação de um SDW [4]. Por natureza, é um sistema bastante complexo, que levanta imensas dificuldades e problemas ao longo da realização de cada uma das fases do tradicional ciclo de desenvolvimento de um SDW. Como tal, é clara a importância da realização de uma implementação adequada de qualquer sistema de povoamento para a garantia do sucesso de um SDW. Uma implementação pobre ou ineficiente pode resultar em algo que não garanta a qualidade da informação que usa para fazer o povoamento e, como tal, coloca em causa a validade e a utilidade do próprio SDW [2]. Como sabemos, um sistema de ETL pode ser especificado, e em alguns casos desenvolvido, através da utilização de uma linguagem de *workflowing*, que nos permita expressar tarefas capazes de serem integradas nos processos macro de um sistema de ETL. Esses processos assentam, normalmente, em três grandes classes de componentes, nomeadamente: angariação, transformação e carregamento de dados. Na prática, isto

significa que qualquer linguagem para ETL deverá ter mecanismos e componentes que nos permitam realizar os mais diversos tipos de operações dentro de cada uma das classes referidas, bem como permitir definir adequadamente a forma como tais componentes coordenam as atividades que desenvolvem entre si. Neste trabalho, enveredámos por uma vertente de desenvolvimento orientada especificamente para a redução da "distância" que existe frequentemente entre fase de modelação conceptual e a fase de apresentação de um modelo físico, mesmo sabendo que não conseguimos, à partida, num modelo conceptual, refletir todos os aspetos físicos que uma verdadeira implementação de um sistema de ETL considera. Assim, para demonstrarmos uma contribuição prática para essa "eliminação", decidimos produzir a partir de uma especificação em *Redes de Petri Coloridas* (RPC) uma primeira versão física do sistema modelado, algo que posteriormente sabemos precisar de ser complementado e ajustado aos requisitos operacionais do sistema de ETL. A esse primeiro esquema físico atribuímos a designação de "esqueleto" ETL, com o intuito de nos referirmos apenas a uma estrutura de suporte elementar. Após uma breve apresentação das RPC e da sua aplicação, utilizando um dos padrões de ETL que desenhámos e modelámos previamente (secção 2), apresentamos o processo de geração dos esqueletos para sistemas ETL (secção 3), revelando a forma como estes podem ser importados e utilizados numa ferramenta de desenvolvimento de sistemas de ETL. Terminamos, depois, este artigo, com algumas breves conclusões.

2 RPC e Padrões ETL

A aplicação de RPC [3] à especificação de padrões de sistemas de ETL, em particular, é algo que tem vindo a ser desenvolvido com algum sucesso. Na terminologia dos SDW, um padrão ETL é um elemento de trabalho que representa uma ou mais tarefas consideradas comuns na implementação de um sistema de ETL. Estes elementos são formalizados usualmente através de um conjunto de processos ou tarefas, recorrendo a um conjunto de atividades pré-estabelecidas [5]. Dois dos padrões mais utilizados em sistemas de ETL - *surrogate key pipelining* e *change data capture* - foram modelados e validados anteriormente em [6], respetivamente, recorrendo a modelos especificados com RPC. O uso desta linguagem para a modelação de padrões de sistemas de ETL permitiu avaliar e validar se um dado modelo reflete *a priori* o tipo de comportamento que se espera que o sistema que representa tenha no futuro. Acreditamos que, tal permite evitar intervenções mais sérias durante a fase de implementação e consequentemente reduzir gastos com recursos computacionais não previstos ou extraordinários. De forma a podermos explicar como a modelação de padrões pode ser executada com RPC, selecionámos um outro padrão de ETL: a dimensão de variação lenta - *slowly changing dimension* (SCD) - com manutenção de história [7]. Este padrão foi projetado e modelado previamente com RPC (fig. 1) de forma a ser capaz de anotar as diversas alterações que vão ocorrendo ao longo do tempo nos dados contidos nos sistemas de informação alvo e que são responsáveis pela alimentação das tabelas de dimensão do SDW. No modelo RPC do padrão ETL SCD, o primeiro passo que se realiza tem a ver com a obtenção de todos os registos que estão na tabela de auditoria relativa à tabela dimensão envolvida - *Audit Records*. Em seguida, os registos recolhidos são

submetidos a um processo de validação assegurado pela transição *Audit Data Verification*. Esta transição é um apontador para a página *Data Verification*, que é a página responsável por verificar, registo a registo, se os dados recolhidos estão de acordo com as regras de negócio estabelecidas na regulação do funcionamento do SDW.

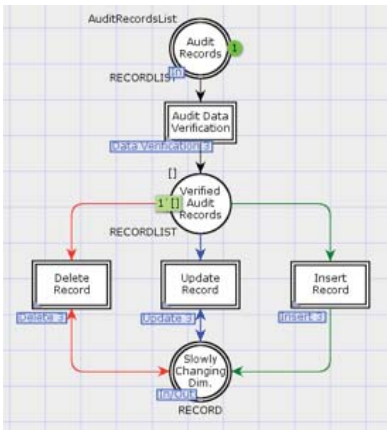


Fig. 1. Modelo RPC para o padrão ETL SCD – figura extraída de [7].

Caso esses dados não apresentem qualquer problema para o processo em causa, eles serão encaminhados para o lugar *Verified Audit Records*. Caso contrário, os registos serão colocadas em tabelas de erro específicas. Mas tarde, os registos que não passaram, por algum motivo, o filtro de validação, serão analisados e tratados por alguém responsável por essa tarefa. Passada esta primeira fase de tratamento, temos um conjunto de registos devidamente verificados no lugar *Verified Audit Records*.

3 O Processo de Geração de Esqueletos ETL

A exportação do modelo RPC apresentado pode ser realizada através do ambiente de trabalho da ferramenta CPN Tools [1]. Este é processo bastante simples que pode ser realizado a partir da área índice da CPN Tools, que nos permite, basicamente, guardar o modelo desenvolvido nos formatos *Petri Net Markup Language* (PNML) ou *Extensible Markup Language* (XML). No nosso caso, optámos pelo segundo formato, uma vez que nos permite descrever parcialmente o comportamento de programas que são capazes de os processar e porque segue a norma ISO/IEC 15909. Na fig.2 podemos ver um pequeno excerto do ficheiro XML que foi gerado para o nosso exemplo de trabalho. O processo relativo à criação e implementação de um esqueleto ETL apenas terminará quando este for importado para o ambiente de trabalho de uma ferramenta de construção de sistemas de ETL. Para o nosso caso de demonstração e acolhimento dos esqueletos modelados escolhemos: o *Pentaho Data Integration* (PDI), uma ferramenta *open-source* que não requer a aquisição de uma licença para a sua utilização, podendo ser instalada em várias plataformas computacionais, como o Windows, o Linux, ou Mac OSX. No caso do padrão ETL SCD que foi modelado, o

resultado final foi uma tradução direta do modelo RPC. Cada uma das quatro transições existentes na página principal do modelo foram traduzidas diretamente num componente ETL do PDI, denominado *Execute Transformation*, que é o componente responsável por executar uma dada transformação. Assim, a geração do esqueleto ETL traduziu-se em quatro *Execute Transformation* componentes, denominados respetivamente por *Audit Data Verification*, *Delete Records*, *Insert Records* e *Update Records*.

```
<page id="ID1422273702">
    <pageattr name="SCD/H 3"/>
    -- Representação XML para o lugar Verified Audit Records.
    <place id="ID1422273978">
      (...)
      <text>Verified Audit Records</text>
      (...)
    </place>
    -- Representação XML para o transição Insert Record.
    <trans explicit="false" id="ID1422273704">
      ( ... )
      <text>Insert Record</text>
      ( ... )
      -- Representação XML para a referência da transição a outra página.
      <subst portsock="(ID1422273769,ID1422273978)(ID1422273765,ID1422273981)"
subpage="ID1422273710">
      ( ... )
      </trans>
      ( ... )
</page>
```

Fig. 2. Fragmentos de um ficheiro XML com a especificação de um modelo RPC.

No formato XML os componentes são identificados pelo elemento XML *step* (fig. 3). Para diferenciar cada um dos diferentes componentes devemos utilizar o elemento XML *type*, que no caso de uma tabela de entrada tem o texto *TableInput* e numa tabela de saída o texto *TableOutput*. No caso do componente de validação, o elemento *type* tem o texto *Validator*. A ligação entre diferentes componentes ETL é realizada através do elemento XML *order*. Uma ligação é um *hop* com elementos *from* e *to*, que correspondem, respetivamente, ao *step* origem e ao *step* destino. No caso das ligações, estas não são identificadas pelo tipo do *step*, mas sim pelo seu nome. Logo não poderá haver nomes de componentes ETL repetidos. Esta é uma regra que qualquer esqueleto ETL tem que necessariamente de obedecer. No caso particular da transformação *Delete Records* existe uma tabela de entrada, que contém os registos verificados durante a transformação *Audit Data Verification*, três tabelas de saída e um componente de seleção. Quanto às tabelas de entrada e de saída a sua representação em XML já foi apresentada. Mas, ainda falta a componente de seleção. Esta é identificada pelo elemento *type* com o texto *SwitchCase*. Para a transformação *Update Records* já definimos todas as representações XML dos seus componentes, não havendo, assim, a necessidade de os detalhar. Quanto ao processo de importação este é bastante simples. Através da opção *File* do menu principal da ferramenta PDI, acedemos à opção *Import from an XML file* e identificamos o ficheiro relativo ao esqueleto ETL que queremos importar. Depois de confirmar, obtemos o resultado que

podemos observar na fig. 4. Nela podemos identificar as quatro tarefas principais anteriormente modeladas, que aqui estão representadas por quatro componentes *Transformation Executer* do PDI.

```
-- Representação XML para o step Audit Data Verification.
<step>
        <name>Audit Data Verification</name>
        <type>TransExecutor</type>
        ( ... )
        <specification_method>filename</specification_method>
        <filename>/Users/hmg/Desktop/auditDataVerification.ktr</filename>
        ( ... )
</step>
-- Representação XML para o step DimCustomer.
<step>
    <name>DimCustomer</name>
    <type>TableOutput</type>
    ( ... )
</step>
```

Fig. 3. Pequeno excerto de um ficheiro XML relativo a alguns *steps* incluídos na modelo físico do esqueleto importado.

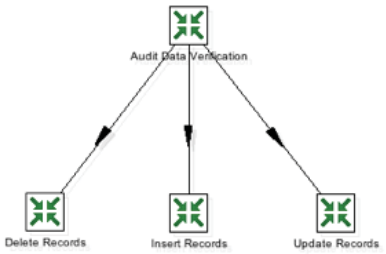


Fig. 4. O esquema físico geral do padrão ETL SCD em PDI.

De seguida, na fig. 5 podemos ver duas outras representações mais detalhadas, cada uma delas correspondendo a dois *Transformation Executers* integrados no esquema físico geral do esqueleto ETL em SCD, representando, por sua vez, duas das tarefas que foram previamente modeladas também com RPC - *Audit Data Verification* e *Insert Record*, respetivamente. De referir que, por falta de espaço, os modelos RPC destas duas tarefas não foram incluídos neste trabalho.

6 Conclusões e Trabalho Futuro

Depois da realização deste trabalho, não hesitamos em dizer que a geração de esqueletos ETL a partir de uma dada ferramenta de especificação (como as RPC) para posterior importação numa ferramenta de implementação de sistemas de ETL é uma

clara mais valia, sobretudo porque dá um outro significado e importância ao modelo e ao próprio processo de modelação de um sistema de ETL, além de adiantar algum serviço de implementação de uma forma praticamente automática. Diríamos mesmo que, com a geração adequada e validação de um modelo RPC para um sistema ETL (ou de um padrão ETL como foi o nosso caso), a estrutura base do modelo físico correspondente à sua implementação praticamente está assegurada. Neste artigo, utilizámos apenas como caso de estudo um único padrão ETL, mas isso foi suficiente para demonstrar como uma especificação conceptual (ou lógica) pode ser diretamente traduzida num modelo físico passível de ser executado.

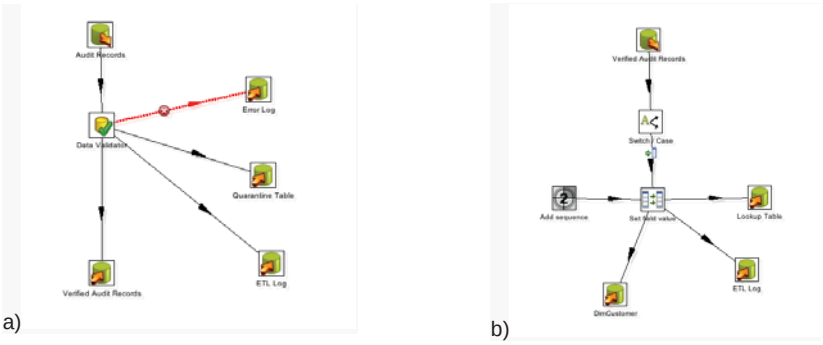


Fig. 5. Esquemas físicos das atividades Audit Data Verification (a) e Insert Record (b) em PDI.

Referências

[1] CPN Tools, 2014. [Online] Availale at <http://cpntools.org/> [Accessed on 28 April 2014].

[2] English, L. P., Improving data warehouse and business information quality: methods for reducing costs and increasing profits. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[3] Jensen, K., Kristensen, M., Wells, L., Coloured petri nets and cpn tools for modelling and validation of concurrent systems. Int. J. Softw. Tools Technol. Transf. 9, 213–254, 2007.

[4] R. Kimball and J. Caserta, The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. 2004, p. 528.

[5] Oliveira, B. & Belo, O., ETL Standard Processes Modelling: A Novel BPMN Approach. In 15th International Conference on Enterprise Information Systems (ICEIS), 2013.

[6] Silva, D., Fernandes, J.M., Belo, O., "Assisting Data Warehousing Populating Processes Design Through Simulation Using Coloured Petri Nets", In Proceedings of 3rd Industrial Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH' 2013), Reykjavik, Iceland, July 29-31, 2013.

[7] Silva, D., Fernandes, J.M., Belo, O., "Approaching ETL Processes Modelling with Coloured Petri Nets", Technical Report, Department of Informatics, University of Minho, 2013.

Modelado multidimensional para la visualización integrada de Big Data en plataformas de Inteligencia de Negocio

Roberto Tardío[1], Elisa de Gregorio[1], Alejandro Maté[1],
Rafa Muñoz-Terol[1], David Gil[1,2], Hector Llorens[1] and Juan Trujillo[1]

[1,2] Lucentia Research Group
[1] Departamento de Lenguajes y Sistemas Informáticos
[2] Departamento de Tecnología Informática y Computación
Universidad de Alicante, España

{rtardio,edg12,amate,rafamt,hllorens,jtrujillo}@ @dlsi.ua.es
dgil@dtic.ua.es

Resumen. La gran cantidad de información disponible y así como su heterogeneidad han sobrepasado la capacidad de las tecnologías actuales de gestión de datos. El tratamiento con grandes volúmenes de datos estructurados y no estructurados, a menudo referido como Big Data, es un tema de investigación de actualidad así como un importante desafío tecnológico. En este artículo, se presenta un enfoque con el objetivo de permitir consultas OLAP a través de diferentes y heterogéneos orígenes de datos asistidos con herramientas de visualización que faciliten el tratamiento de los mismos. Nuestro enfoque está basado en el paradigma MapReduce, permitiendo la integración de diferentes formatos como el novedoso formato RDF Data Cube. Las principales contribuciones de nuestro enfoque son la capacidad de consultar y visualizar distintas fuentes de información, manteniendo al mismo tiempo, una visión integrada y completa de los datos disponibles, así como una sencilla interfaz de visualización de Big Data. El presente artículo también analiza las ventajas y desventajas, así como los retos de implementación que presenta este enfoque y, finaliza con un caso de estudio mostrando las ventajas de la aproximación presentada.

1 Introducción

En la actualidad, cada vez hay más información presente en Internet que sirve de ayuda en la toma de decisiones. Sin embargo, el tamaño de la información producida al año supera ya los zettabytes [14]. Es imposible procesar, almacenar y gestionar toda esta información en servidores locales, incluso para los sistemas de las principales compañías de Inteligencia de Negocio (IN). La posibilidad de llevar a cabo consultas OLAP (OLAP - On-Line Analytical Processing) sobre enormes cantidades de información y que al mismo tiempo fuesen capaces de recuperar sólo la información relevante en cada momento, proporcionaría im-

portantes beneficios. Sin embargo, dada la heterogeneidad y el tamaño de los datos utilizados, el esfuerzo necesario para aprovechar el valor potencial de toda esta información está al alcance de muy pocas empresas. A menudo la literatura existente [3] se refiere a este fenómeno como Big Data.

El objetivo de este artículo es proponer y analizar un enfoque integrado para permitir consultas OLAP sobre fuentes de datos heterogéneas, donde cada fuente de datos puede contener diferentes dimensiones internas y externas. Se propone un enfoque basado en la estrategia de MapReduce [4] capaz de dividir una consulta, y distribuirla a los diferentes nodos que tienen acceso a diferentes bases de datos de diferentes formatos. La salida de estos nodos se integra posteriormente, haciendo que el proceso sea transparente para el usuario final. Con el fin de aumentar la capacidad de extensión de nuestra propuesta, basamos nuestro enfoque en estándares utilizando SPARQL como el lenguaje de consulta para el módulo de distribución e integración así como entrada para los nodos.

Existen diferentes tecnologías relacionadas con nuestra propuesta. En el área de Big Data y arquitecturas cabe destacar MapReduce [12] como la arquitectura distribuida comúnmente aplicada. En relación a Linked Data destacamos los dos estándares: RDF [6] cuyos elementos están representados por identificadores uniformes de recursos (URI) [10] y SPARQL con su nueva característica conocida como federated queries [2].

El resto del trabajo se ha estructurado de la siguiente manera. En la siguiente sección, se presenta nuestra propuesta de un modelo integrado para acceder a grandes volúmenes de datos. Finalmente, en las dos últimas secciones, se incluye un análisis de las ventajas del modelo y de las dificultades relacionadas con su ejecución, así como futuras direcciones de esta investigación.

2 Un modelo integrado para acceder a Big Data

En esta sección describimos nuestra propuesta, la cual permite a los usuarios el acceso a estructuras de tipo Big Data de manera transparente, al mismo tiempo que se preserva la estructura original de la información. Como se mencionó en la Sección 1, Big Data se caracteriza por la enorme cantidad de información heterogénea, procedente de distintas fuentes, de la que se dispone. Estas enormes cantidades de datos presentan dos problemas principales con el fin de incluirse en una estructura DW tradicional: (i) el alto coste de las consultas y del mantenimiento de la información [11] y (ii) el problema de que algunas de las relaciones semánticas no pueden representarse en el esquema MD tradicional [1, 7], como es el caso de las relaciones conceptuales en ontologías [5]. Por lo tanto, al no poder ser capturadas tales relaciones, se pierden valiosa información y capacidades de análisis.

Con el fin de superar estos problemas, nuestra propuesta distribuye el esquema multidimensional y la responsabilidad de proporcionar la información entre los diferentes nodos que constituyen una red, como se puede ver en la Figura 1. Cada nodo es responsable de la interpretación de las consultas que recibe así como de la administración de sus propios datos. Finalmente, el nodo envía los

datos solicitados de vuelta al módulo central, que integra estos datos de acuerdo con el esquema universal, usando el paradigma de MapReduce. Una vez que el resultado de la consulta se ha integrado en el esquema universal, se presenta al usuario de la misma manera que en un enfoque tradicional.

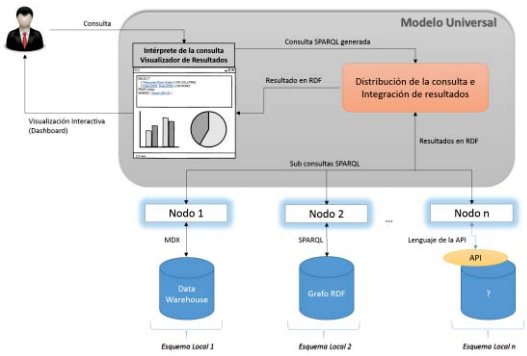


Fig. 1. Visión general de la arquitectura propuesta

A diferencia de modelado MD tradicional, donde toda la estructura del cubo se conoce de antemano, nuestra propuesta permite al usuario consultar cierta información que no aparece en el Esquema Universal. Esto se hace por medio de la delegación de resolución de consultas (query), donde ciertas partes de la consulta se resuelven completamente por un único nodo específico, y los resultados después se integran en el Esquema MD Universal presentado al usuario.

2.1 Extensión al Modelo Multidimensional.

Tradicionalmente, los principales elementos que intervienen en el modelado MD [8] son los siguientes: (i) Los hechos (facts), por ejemplo, las acciones, que son el centro del análisis, (ii) Las dimensiones (dimensions), por ejemplo, la empresa, que representa el contexto de análisis, (iii) Bases, por ejemplo, índice, dentro de la dimensión empresa, que constituyen niveles de agregación en una dimensión dada, (iv) Descriptores (descriptors), por ejemplo, CompanyCode, que constituyen atributos que diferencian instancias en el mismo nivel base, (V) atributos de dimensión (dimension attributes), por ejemplo, CompanyName, que proporcionan información descriptiva adicional, y (vi) los hechos (medidas, “fact attributes”), por ejemplo, Valor, que proporcionan información sobre el rendimiento del proceso que se está analizando. Con la única excepción de los hechos, cualquiera de estos elementos puede considerarse externo. Los hechos no pueden incluirse como elementos externos, ya que definen qué dimensiones y hechos están involucrados en el análisis, así como la granularidad de la tupla.

3 Caso de estudio

En nuestro Caso de Estudio queremos tomar decisiones informadas con respecto a la bolsa de valores. Para esta finalidad, hemos construido un almacén de datos que almacena información sobre las empresas y los índices en los distintos mercados de valores de todo el mundo. Nuestro objetivo es proporcionar respuestas que integran la información del almacén de datos, así como información de otras fuentes externas.

Con el fin de plantear las consultas en un escenario distribuido y heterogéneo, nuestra arquitectura se basa en tres elementos principales: interfaz gráfica de usuario, MapReducer (el módulo de Distribución e Integración), y Nodos Especializados que interactúan con determinados tipos de conjuntos de datos. Un rasgo característico de nuestro modelo es que utiliza los estándares SPARQL y RDF para la integración de datos heterogéneos utilizando una semántica común. A continuación, se describe con más detalle cada parte de la arquitectura, y se muestra su funcionalidad dentro del Caso de Estudio.

Para nuestro estudio de caso consideramos una arquitectura que consta de 3 nodos. Nodo local: almacenamiento de datos basado en Pentaho [9] y Mondrian OLAP Server que contiene precios de las acciones históricas de las empresas de algunos índices específicos del país: IBEX35, DAX, FTSE, DOW, NASDAQ (203 empresas y 4 países). Tales de datos se han obtenido de YAHOO Finanzas[1]. Nodo externo 1: Nodo final SPARQL de Worldbank[2] que contiene un conjunto de indicadores históricos sobre una amplia gama de países (PIB, población, empleo, IPC, etc). Nodo externo 2: API de búsqueda de Twitter que puede dar el número de opiniones (total, positivo o negativo) introduciendo como término de búsqueda cualquier cadena de texto.

Elegimos como claves comunes para todos estos nodos país y empresa. No se requiere que todos los nodos tengan todas las claves. A continuación, tenemos el módulo de interfaz gráfica de usuario que transforma las consultas de entrada del usuario en una consulta SPARQL válida para el módulo Distribución e Integración.

Para este Caso de Estudio, vamos a utilizar la consulta de ejemplo "¿Cuál es la diferencia en comparación con hace un año en el precio de la acción de las empresas que operan en países cuyo PIB creció más de un 1% y que tengan más de 200 opiniones en Twitter?

El módulo de Distribución e Integración es el responsable de dividir la consulta de entrada de la interfaz gráfica de usuario en consultas válidas para los nodos correspondientes. Para realizar esta tarea, el módulo de Distribución e Integración hace uso del modelo de implementación y el esquema universal. Después de que las consultas han sido enviadas, espera sus resultados parciales en formato RDF con el fin de reducirlos al formato RDF Data Cube y devolver el resultado integrado a la interfaz gráfica de usuario.

[1] http://finance.yahoo.com/
[2] http://data.worldbank.org/

Con el fin de distribuir la consulta, en primer lugar, el módulo de proceso de distribución analiza la consulta original SPARQL y, utilizando el modelo de implementación (nodo-tipo-*mdelement), identifica qué partes de la consulta deben enviarse a cada nodo. Durante este proceso, se detecta si las consultas se pueden resolver en paralelo o existen dependencias entre ellas y, por lo tanto, tienen que ser secuenciales.

Los resultados obtenidos por los diferentes nodos son enviados de vuelta al integrador en forma de triples RDF, donde se pueden integrar de acuerdo con el esquema universal de por medio de operaciones algebraicas básicas (intersección, unión o resta).

Los nodos actúan como puentes intermediarios entre el modelo conceptual y los diferentes tipos de fuentes de datos, cada una con sus particularidades. Por un lado, los nodos locales contienen normalmente una gran cantidad de datos privados altamente multidimensionales, y por regla general no se almacenan en RDF, sino más bien en un formato más eficiente y estructurado (por ejemplo, arquitecturas de bases de datos multidimensionales para consultar a través de plataformas de BI). Por otra parte, las fuentes de datos externas serán idealmente nodos finales RDF.

Una vez de que los resultados han sido fusionados, se devuelven a la interfaz gráfica de usuario donde se visualizan por medio de un panel en el cual se emplean diferentes tecnologías de visualización tales como Google Charts [13], etc.

4 Discusión y trabajo futuro

En este trabajo hemos presentado un enfoque que permite realizar consultas heterogéneas de fuentes Big Data y de una manera integrada. Esto se hace mediante el uso de nodos independientes, que se reparten el manejo de distintas fuentes de datos locales y externas. En nuestro enfoque, cada nodo que proporciona información mantiene su estructura interna, por lo tanto no hay pérdida de información. Las principales ventajas de nuestro enfoque son que (i) se proporciona una visión unificada de los datos, lo que permite añadir y eliminar nodos e información de forma sencilla y transparente, y (ii) mantiene la estructura de la información en cada nodo, por lo tanto se conserva la rica semántica de relaciones y puede ser consultada incluso si esta información no está presente en el esquema universal.

Las consultas son más rápidas, pero los datos que se transmiten y se integran después son mayores. Esta estrategia tiene dos ventajas principales si una consulta implica muchos nodos: (i) no hay necesidad de esperar a las respuestas de nodo antes de enviar todas las consultas, y (ii) la integración se puede hacer gradualmente a medida que las respuestas son recibidas por el integrador.

El principal trabajo futuro es llevar a cabo una implementación eficiente del modelo y evaluarlo. La optimización del proceso de consulta puede resultar en una herramienta de consulta de gran alcance para integrar datos locales con información adicional proveniente de la web, enriqueciendo así el análisis OLAP actual. Con el fin de probar la aplicabilidad de nuestro enfoque, el modelo se

pondrá a prueba en empresas reales, cuyos datos serán privados, pero con el modelo presentado se vinculará a un conjunto de fuentes de datos abiertos públicos, mejorando así la capacidad OLAP la empresa.

Por último, consideramos mejorar la interfaz con un módulo de procesamiento de lenguaje natural que actúa como un sistema de búsqueda de respuestas en el que las consultas se pueden introducir en un idioma inglés controlado, por lo que es más fácil la consulta de los datos y se evita la necesidad de especificar complejas consultas SPARQL.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad español bajo el proyecto GEODAS-BI (TIN2012-37493-C03-03).

Referencias

1. Abelló, A., Samos J., & Saltor, F. (2006). YAM2: a multidimensional conceptual model extending UML. *Information Systems*, 31(6), 541-567.
2. Buil-Aranda, C., Arenas, M., Corcho, O., & Polleres, A. (2013). Federating queries in sparql 1.1: Syntax, semantics and evaluation. *Web Semant.*, 18(1):1– 17.
3. Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J., & Welton, C. (2009). Mad skills: new analysis practices for big data. *Proceedings of the VLDB Endowment,* 2(2):1481–1492.
4. Dean, J. & Ghemawat, S. (2008). MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.
5. Jacobs, A. (2009). The pathologies of big data. *Communications of the ACM*, 52(8):36–44.
6. Klyne, G., Carroll, J., & McBride, B. (2004). Resource description framework (rdf): Concepts and abstract syntax. *W3C recommendation*, 10.
7. Luján-Mora, S. and Trujillo, J. (2006). Physical modeling of data warehouses using uml component and deployment diagrams: design and implementation issues. *Journal of Database Management (JDM)*, 17(2):12–42.
8. Luján-Mora, S., Trujillo, J., & Song, I. (2006): A uml profile for multidimensional modeling in data warehouses. *Data & Knowledge Engineering*, 59(3):725–769.
9. Marinheiro, A. & Bernardino, J. (2013). Analysis of open source business intelligence suites. In *Information Systems and Technologies (CISTI), 2013 8th Iberian Conference on*, pages 1–7.
10. Masinter, L., Berners-Lee, T., & Fielding, R. (2005). Uniform resource identifier (uri): Generic syntax.
11. Niemi, T., Niinimäki, M., Nummenmaa, J., & Thanisch, P. (2002). Constructing an olap cube from distributed xml data. In *Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP*, pages 22–27. ACM.
12. White, T. (Ed.). (2009). Hadoop: Definition guide. Sebastopol, CA (United States): O'Reilly Media, Inc.
13. Ying, Z. (2012). Introducing google chart tools and Google Maps api in data visualization courses. *IEEE Computer Graphics and Applications*, 32:6–9.
14. Zikopoulos, P. & Eaton, C. (2011). Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. *McGraw-Hill Osborne Media*, 1st edition.

Bioqueries: a Social Community for SPARQL queries in Life Sciencess

María Jesús García Godoy, Esteban López-Camacho, Ismael Navas-Delgado, and José F. Aldana-Montes*

Lenguajes y Ciencias de la Computación, Universidad de Málaga, Bulevar Louis Pasteur 35, Málaga, Spain
{mjgarciag,esteban,ismael,jfam}@lcc.uma.es

Abstract. Life Sciences have emerged as a key domain in the Linked Data community because of the diversity of data semantics and formats available through a great variety of databases and web technologies. Unfortunately, bioinformaticians are not exploiting the full potential of this technology and experts in Life Sciences have real problems to discover, understand and devise how to take advantage of these interlinked data. In this context, we have implemented Bioqueries, a wiki-based portal that is aimed at community building around biological Linked Data (`http://bioqueries.uma.es/`). This space offers a collaborative platform in which users can create, modify, execute and share biological SPARQL queries.

1 Introduction

The amount of biological information has increased over the last few years. The way that we share and access this information has been altered by the lowering of the barriers to accessing this data. However, despite the accessibility of such biological data, the application developers have difficulties in reusing this information. Therefore, in an effort to address this problem, the Linked Data (LD) technology has emerged. LD is defined as a set of best practices (using standards) with the objective of connecting, sharing and exposing data and knowledge. In the Life Sciences domain, the LD tools are not targeted at biologists because they do not take into account end-user usability. Therefore, there have been some attempts to improve the application of this technology to the biological domain such as the EBI RDF platform [2].

Bioqueries aims to start the process towards a greater understanding of Life Sciences LD sources through a collaborative environment based on social networks [1]. Bioqueries opens up a way to build-up communities around a shared interest in certain biological domains using public LD. This is achieved by sharing a virtual space for the design and execution of (federated and non-federated) SPARQL queries that are executed and documented using natural language descriptions. Since 2011, the Bioqueries platform has grown to contain 279 query

* Corresponding author

entries (27 out of them are federated), more than 1000 registered users and 60 endpoints.

2 Bioqueries platform

In this section, the Bioqueries features implemented are listed as follows:

- Bioqueries provides a SPARQL registry in which users can suggest and add new endpoints that will be checked by users with administrator privileges. In order to interrogate each endpoint, Bioqueries includes a query registry for federated and non-federated SPARQL queries.
- Parameterized SPARQL query execution allows users to retrieve information introducing values for the query parameters. The results are shown in HTML, RDF/XML, N3 and graph.
- Bioqueries navigation is also available. A query explorer allows the queries in a list to be organized according to different topics. The portal provides a query search engine with an advanced search option. Bioqueries content includes a set of 279 query entries that have been increasing since the initial 112 queries. These queries are classified under different biological categories.
- An interesting Bioqueries feature is the federated SPARQL queries which retrieve information from multiple distributed endpoints. The information retrieved is useful for taking full advantage of the LD in the biological domain.

Bioqueries is a social platform for sharing and executing SPARQL queries and tries to bring end-users with biological or bioinformatic backgrounds closer to LD technology. Registered users can design queries from scratch or using the clone option to edit any query that has been already registered. Users can exchange opinions, learn and obtain results from SPARQL queries for their research field. As a result of the success of the Bioqueries social community, new features are being implemented to improve the user experience.

3 Acknowledgement

Funded by Grants TIN2011-25840 (Ministerio de Ciencia e Innovación) and P11-TIC-7529 and P12-TIC-1519 (Plan Andaluz de Investigación, Desarrollo e Innovación).

References

1. García-Godoy, M.J., López-Camacho, E., Navas-Delgado, I., Aldana-Montes, J.F.: Sharing and executing linked data queries in a collaborative environment. Bioinformatics 29(13), 1663–1670 (2013)
2. Jupp, S., Malone, J., Bolleman, J., Brandizi, M., Davies, M., Garcia, L., Gaulton, A., Gehant, S., Laibe, C., Redaschi, N., Wimalaratne, S.M., Martin, M., Le Novère, N., Parkinson, H., Birney, E., Jenkinson, A.M.: The EBI RDF Platform: Linked Open Data for the Life Sciences. Bioinformatics 30(9), 1338–1339 (2014)

Adding Semantic Modules to improve Goal-Oriented Analysis of Data Warehouses using I-star

Alejandro Maté[1], Juan Trujillo[1], and Xavier Franch[2]

[1] Lucentia Research Group, Department of Software and Computing Systems, University of Alicante, Spain
{amate,jtrujillo}@dlsi.ua.es
[2] BarcelonaTech, Universitat Politècnica de Catalunya
franch@essi.upc.edu

Abstract. Requirements elicitation and analysis is a key step in designing and maintaining data warehouses. In order to better support this step, in this paper we (i) propose an extension of the basic goal-oriented metamodel in order to include semantic modules, (ii) include a description of each step followed in the process, and (iii) evaluate it by means of an empirical experiment.

Keywords: Data warehouses, user requirements, i-star

1 Summary

Data warehouses (DW) are a cornerstone in Business Intelligence systems, since they support the decision making process by storing the historical data that is used in order to make more informed decisions. The development of the DW is a complex process which must be carefully planned in order to meet users' needs. This process can become even more complex, if we consider that requirements for the DW change as the organization's information needs change. For this reason, modeling user requirements is a very important aspect of DWs.

Recent works focus on representing DW requirements in terms of goals [3, 2, 4]. Among the different goal-oriented proposals, the i* framework [5], is currently one of the most widespread goal modeling frameworks. This framework has been applied for modeling organizations and system requirements among others.

Nevertheless, as pointed in [1], the i* framework lacks scalability due to the absence of modularity. As DW requirements models may become very complex, this hurts their readability and comprehension, becoming more difficult to correct and update as the requirements change. Therefore, it is important to improve this aspect in order to adequately manage modifications in DW requirements.

In order to tackle this problem, in this paper we propose an ontology mapping from i* to the DW domain, and consider the inclusion of modules in order to address its lack of scalability for DWs. In turn, this increases its manageability, as well as the comprehension capability of the user when dealing with

complex models. With these modifications, the communication between users and developers is improved, leading to higher success rates. We also provide a set of guidelines to correctly apply the proposal and assess the validity of the proposal by means of an empirical experiment.

In order to assess the validity of the proposal, we have performed two experiments, with participants ranging from non-expert people to DW designers and experts on i* modeling. In our first experiment, we analyze how participants perform when identifying elements and what characteristics they perceive in the models. In our second experiment, we include modification tasks over existing models, as well as the creation of a new model. To prevent the order of the questions from affecting the results, we created several types of questionnaires for each experiment, that were distributed equally among the participants. Finally, we collected the results and perfomed an ANOVA analysis in order to identify strong correlations between the structure of the model and the results obtained.

The results obtained show that the time spent was notably superior for models created with a modularized approach. However, the time spent per element drawn was inferior when the model was modularized. Most importantly, the average number of elements identified from the text which described the model was superior when modules were applied, as opposed to the monolithic structure. Additionally, some monolithic models, presented repeated elements, which should not be created, and tend to increase in number as the model gets bigger.

As future work we plan to adapt the proposal to evaluate how the substitution of the intermediate abstraction level (information) by a lower abstraction level (information requirement) affects the definition of the requirements models. This way, both DW designers and users can focus strictly either on analyzing the validity of the current set of goals or on the information to be stored in order to make decisions.

Acknowledgments. This work has been partially supported by the ProS-Req (TIN2010-19130-C02-01) and by the MESOLAP (TIN2010-14860) and SERENIDAD (PEII-11-0327-7035) projects from the Spanish Ministry of Education and the Junta de Comunidades de Castilla La Mancha respectively. Alejandro Maté is funded by the Generalitat Valenciana under an ACIF grant (ACIF/2010/298).

References

1. Franch, X.: Incorporating Modules into the i* Framework. In: CAiSE, LNCS, vol. 6051, pp. 439–454. Springer Berlin (2010)
2. Giorgini, P., Rizzi, S., Garzetti, M.: Grand: A goal-oriented approach to requirement analysis in data warehouses. Decision Support Systems 45(1), 4–21 (2008)
3. Mazón, J.N., Pardillo, J., Trujillo, J.: A model-driven goal-oriented requirement engineering approach for data warehouses. pp. 255–264. ER'07, Springer-Verlag (2007)
4. Prakash, N., Gosain, A.: An approach to engineering the requirements of data warehouses. Requirements Engineering 13(1), 49–72 (2008)
5. Yu, E.S.K.: Modelling strategic relationships for process reengineering. Ph.D. thesis, Toronto, Ont., Canada, Canada (1995)

Modelado de calidad de contexto con MLContext

José R. Hoyos[1], Jesús García-Molina[1], Juan A. Botía[1,2]

[1]Universidad de Murcia, Facultad de Informática, España
{jose.hoyos, jmolina, juanbot}@um.es
[2]Middlesex University, London, UK
J.BotiaBlaya@mdx.ac.uk

Resumen. Un aspecto crucial de los sistemas context-aware es la calidad de la información de contexto obtenida de diversas fuentes. Esta información puede ser incompleta, imprecisa o ambigua. Por ello debe ir acompañada de otra información que describa su calidad (QoC, Quality of Context) con el fin de determinar si puede ser usada. MLContext es un lenguaje específico del dominio para el modelado del contexto que ha sido creado siguiendo los principios de MDE (Model-Driven Engineering). En este artículo se describe una extensión de MLContext para modelar QoC por medio de la definición de parámetros de calidad de los sensores. Se presenta un escenario y se describe cómo crear su modelo del contexto teniendo en cuenta la calidad. El modelo creado es validado con y sin parámetros de calidad, comparando los resultados y ofreciendo las conclusiones sobre el enfoque planteado.

Palabras clave: Modelado del Contexto, Calidad del contexto, Context-aware, Ingeniería de Software Dirigida por Modelos

1 Introducción

Los servicios context-aware utilizan la información del contexto para adaptar su comportamiento o el contenido que ofrecen a los usuarios, por lo que los manejadores de contexto son un elemento esencial en la arquitectura de estos servicios. En la última década se ha realizado un esfuerzo importante en el ámbito del modelado del contexto [13] y la implementación de manejadores de contexto [14], sin embargo "todavía existen pocos modelos y herramientas que ayuden a diseñadores y desarrolladores a dominar la complejidad de la información del contexto" [15].

Como ha sucedido en otros dominios, la aplicación de técnicas MDE (Model Driven Engineering) en el desarrollo de aplicaciones context-aware ha sido explorada en algunos trabajos [4, 3, 1] con el objetivo de elevar los niveles de abstracción y automatización. En este sentido, MLContext es un lenguaje específico de dominio (DSL) independiente de cualquier plataforma que permite modelar el contexto a un alto nivel de abstracción y a partir de los modelos creados generar automáticamente artefactos usados en middlewares context-aware para el manejo del contexto, tales como ontologías y código Java [1].

La necesidad de considerar la calidad de la información del contexto fue expuesta por primera vez en [5], donde se consideraba que la calidad del contexto (QoC) era toda información que describía la calidad de la información que formaba parte del contexto. Desde entonces, se ha prestado gran atención a la QoC y se han publicado numerosos trabajos que proponen formas de representar, medir y evaluar la QoC, sin embargo pocos manejadores de contexto soportan actualmente QoC [15].

En [2] presentamos una extensión de MLContext para el modelado de parámetros y requisitos de calidad y mostramos cómo era posible la generación automática de código Java relacionado con la evaluación de la QoC. Ahora, se explica en detalle cómo abordar el modelado de la QoC con MLContext a través de la creación de un modelo de contexto para un caso de estudio AAL (Ambient Assisted Living), y se evalúa el código generado a través de una simulación del caso de estudio que permite comprobar cómo afectan los parámetros de calidad a la detección de situaciones de contexto. Además se presentan nuevas características del lenguaje como la parametrización y composición de situaciones y se relaciona el enfoque con otros también basados en modelos. El trabajo se ha organizado de la siguiente forma: en la sección 2 se describe brevemente el lenguaje MLContext; en la sección 3 se indican las etapas en el modelado del contexto y de su calidad; en la sección 4 se describe un caso de estudio AAL, que se modela en la sección 5; en la sección 6 se evalúa el modelo por medio de una simulación y se hace un análisis de los resultados, en la sección 7 se comentan algunos trabajos relacionados y, finalmente en la sección 8 se presentan las conclusiones.

2 MLContext: Un Lenguaje de Modelado de Contexto

La *calidad del contexto* se define como "cualquier información que describe la calidad de la información usada como información de contexto" [5]. Son muchos los criterios de calidad identificados en la literatura y existen numerosas clasificaciones [5,6,7]. Existe también cierta confusión en cuanto al significado que cada autor le asigna a los parámetros, como sucede, por ejemplo, con exactitud y precisión. Un estudio comparativo del uso de parámetros de calidad en varios trabajos se puede encontrar en [8]. Nosotros hemos realizado una revisión de la literatura e identificado los parámetros de calidad más comunes y los hemos clasificado en tres categorías [2]:

- *Adquisición de los datos*: precisión, exactitud, posición, momento de la adquisición, cobertura y rango.
- *Representación de los datos*: unidades, formato, alias, comprensión.
- *Uso de los datos*: credibilidad, completitud, disponibilidad, comparabilidad y relevancia.

MLContext [1] es un lenguaje especialmente diseñado para modelar el contexto de las aplicaciones context-aware. El modelo del contexto es separado del modelo de la lógica de negocio para favorecer la evolución y legibilidad. Se trata de un lenguaje textual que es el elemento principal de una estrategia MDE para generar automáticamente artefactos software relacionados con el manejo de la información de contexto

en middlewares context-aware, como son ontologías OWL, código Java o código GML (Geography Markup Language) en un escenario de GeoSPARQL. MLContext proporciona un nivel alto de abstracción para construir modelos independientes de una plataforma concreta utilizando un lenguaje simple y sencillo de aprender, de manera que el usuario final pueda escribir y entender los modelos de contexto.

Un modelo MLContext incluye entidades y fuentes de contexto. Cada entidad se especifica por medio de un conjunto de informaciones de contexto que se corresponden con los tipos de contexto más comunes y cada fuente de contexto por una interfaz que lo desacopla del dispositivo concreto que proporciona los datos.

En cuanto a la QoC, MLContext permite modelar los atributos de calidad de la información con la que se trabaja y los requisitos de calidad de una aplicación. No sólo es posible expresar parámetros de calidad objetivos (como la precisión de una fuente de contexto), sino también otros subjetivos (como la relevancia de la información de contexto). A partir de la información primaria que suministran los sensores cuando se realiza modelado de contexto es necesario definir información de contexto más abstracta que depende de la aplicación [11], por ejemplo la relación de que una persona está una habitación particular o cerca de otra persona a partir de las coordenadas proporcionadas por un sensor. MLContext proporciona las *situaciones de contexto* para definir contexto abstracto a través de un sencillo lenguaje de expresiones que puede incluir operadores relacionados con la QoC.

En [12] se plantean algunos de los retos que existen en el modelado de la QoC. En el diseño de MLContext hemos procurado dar respuesta a algunos de ellos.

1. *Expresividad de los parámetros de QoC*. Cómo se indicó arriba, MLContext utiliza los parámetros de calidad anteriores con una semántica bien definida.
2. *Parámetros de calidad separados de la información de contexto*. El valor de un parámetro de calidad puede cambiar dinámicamente y muy rápido, haciendo que sea difícil hacer predicciones sobre qué información de contexto tiene mejor calidad. En MLContext se definen separados de la información del contexto y ligados a las fuentes de contexto que suministran la información. De esta forma, como la fuente puede cambiar dinámicamente, se asegura que la información de calidad del contexto esté actualizada.
3. *Decidir la credibilidad de una fuente*. En muchos casos, una aplicación context-aware debe decidir si la información que le suministra una fuente de contexto es fiable. Esto puede realizarse de dos formas: i) comprobando la ausencia de contradicciones, y ii) comprobando su consistencia comparándola con la misma información suministrada por otras fuentes. Nuestro enfoque permite asociar varias fuentes que suministran la misma información, de manera que la aplicación puede comparar los datos que suministran y ver si coinciden. Un modelo MLContext posibilita comprobar si hay contradicciones entre dos fuentes que, sin suministrar exactamente un valor para una misma propiedad, proporcionan sin embargo información relativa al mismo contexto que debe ser consistente. Esto se realiza modelando una situación de contexto, como veremos en nuestro ejemplo para detectar la presencia

de una persona en una habitación mediante la posición que nos devuelve una red ZigBee y la información de un sensor de movimiento en esa habitación.

En la Sección 5 se presenta un ejemplo de modelado de contexto con MLContext que ilustra las características mencionadas, después de explicar en las siguientes secciones cómo modelar el contexto con este lenguaje y proponer el caso de estudio a modelar.

3 Modelado del Contexto con MLContext

La mayoría de enfoques consideran que el modelo de contexto incluye, además de la información de contexto propiamente dicha (es decir la información sobre las entidades de la aplicación), información adicional sobre situaciones de contexto, actividades y detalles de los sensores [4]. Esta información, que es específica de la aplicación, hace que los modelos no se puedan reutilizar para otros escenarios que usen el mismo contexto y sean menos legibles.

Con MLContext promovemos aplicar una separación de aspectos y modelar el contexto por medio de dos modelos: el modelo de contexto y el modelo de aplicación [1]. El *modelo de contexto* describe las entidades del contexto, sus propiedades y las fuentes de contexto y no contiene detalles específicos de una aplicación concreta. Esta última información se incluye en el *modelo de aplicación* que especifica los detalles técnicos de los sensores y las fuentes de contexto, los parámetros de calidad y algunas situaciones y relaciones complejas cuya semántica puede variar de una aplicación a otra. Esta separación de aspectos facilita la reutilización de un modelo de contexto en varias aplicaciones y la evolución del sistema cuando cambia el contexto, dado que el modelo del contexto no se vería afectado por cambios en los dispositivos ni, en general por cambios en el modelo de aplicación. Cabe señalar que la creación de un modelo de aplicación supone la existencia de un modelo de contexto al cual referencia. Otra ventaja destacable es una mejora en la legibilidad del modelo de contexto al eliminar detalles específicos de aplicaciones concretas y por tanto ser más simple.

En el modelo de contexto pueden especificarse relaciones simples (por ejemplo, la relación "contenido en"). Sin embargo, en sistemas complejos pueden aparecer relaciones más complicadas como la relación "estar_cerca_de". El significado de esta relación puede variar dependiendo del dominio de la aplicación y debería especificarse en el modelo de aplicación (como una situación de contexto).

La Figura 1 detalla los pasos que debe seguir el desarrollador para modelar el contexto de una aplicación y que se aplicarán en la siguiente sección a un caso de estudio. Primero se debe crear el modelo de contexto en las siguientes tres etapas:

1. *Modelar las entidades*. Se especifican los nombres y las propiedades de las entidades de la aplicación. Las relaciones simples entre entidades se especifican también mediante propiedades ligadas con otras entidades.
2. *Especificar las categorías*. Se definen las categorías a las que pertenecen las entidades y se asocia cada entidad a una de las categorías. Se pueden definir jerarquías entre categorías. No es necesario definir las propiedades de las categorías, ya que

estas se infieren a partir de las propiedades de sus entidades. Si una entidad no se asigna a una categoría, MLContext genera una para ella de manera automática.

3. *Definir las fuentes de contexto*. Se especifican las fuentes de contexto que suministrarán el valor a las propiedades de las entidades cuyo valor no es una constante. Se pueden definir varias fuentes de contexto para una misma propiedad y que una misma fuente de contexto suministre valores a varias propiedades.

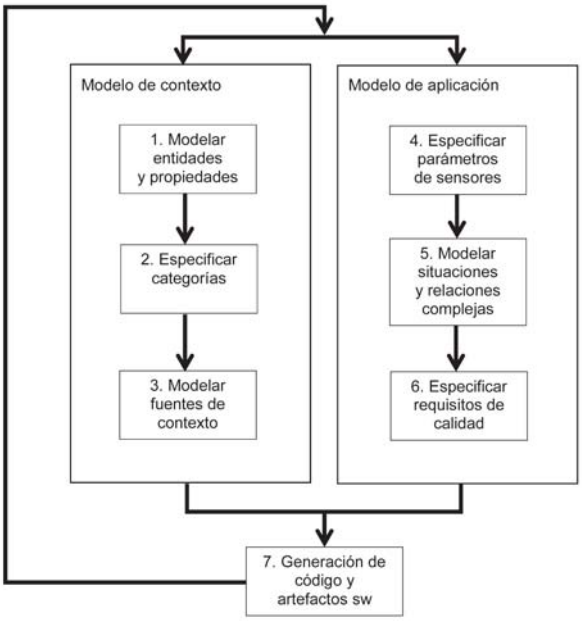


Fig. 1. Etapas en el modelado del contexto con MLContext

Una vez creado un modelo de contexto se crea el modelo de aplicación en tres etapas:

1. *Especificar los detalles técnicos y parámetros de calidad de los sensores*. Se relaciona cada sensor físico con una fuente de contexto.
2. *Especificar algunas relaciones complejas (si las hay) y situaciones de contexto*. MLContext permite agrupar la información de contexto necesaria para determinar si existe una relación entre entidades o si se está produciendo una determinada situación en un instante.
3. *Especificar requisitos de calidad*. Como la información que proporcionan las fuentes de contexto puede ser incierta, MLContext permite especificar en las situaciones algunos requisitos de calidad asociados a la información que servirán más tarde para que el sistema determine si se están produciendo dichas situaciones.

A partir de los modelos de contexto y aplicación se puede generar automáticamente código de la aplicación, y otros artefactos software relacionados con la QoC. Ya existen transformaciones modelo-a-texto predefinidas para generar parte de una ontología OWL, código Java para algunos middlewares y código GPL para GeoSPARQL [1].

Las etapas anteriores se pueden ejecutar dentro de un proceso iterativo en el que las entidades del sistema y fuentes se pueden especificar de forma incremental. El lenguaje permite especificar una fuente de contexto sin ligarla con ninguna entidad, para poder asociarla más tarde cuando se hayan modelado. Este proceso iterativo permite abordar la tarea de implementación y pruebas gradualmente.

MLContext es soportado por una herramienta que incluye un editor[1] que facilita la creación de los modelos, es decir la especificación textual, con facilidades como el resaltado de palabras reservadas, autocompletado de código y una vista *outline* del modelo.

4 Descripción del Caso de Estudio

El siguiente caso de estudio está basado en un escenario AAL de la AAL Open Association (AALOA) [9].

Lisa es una persona de 78 años que vive sola, pero tiene hijos adultos que la cuidan. Aunque viven lejos de Lisa, los hijos pueden saber lo que está haciendo en cada momento a través de la información que les llega de un sistema AAL instalado en la casa de Lisa. En la casa hay instalado un sistema de localización capaz de seguir los movimientos de Lisa y de suministrar su posición a otros subsistemas. Este sistema usa una red ZigBee y otra WIFI, con balizas fijas (que pueden, por ejemplo, usar triangulación para obtener las coordenadas de su posición). Se utilizan dos redes para poder comparar los resultados obtenidos y elegir los más precisos. La exactitud de la red ZigBee es de unos 50 cm, mientras que la de la red WIFI es de 1,50 m, ya que existen muchas más balizas ZigBee por tratarse de una tecnología más barata. La casa dispone también de sensores de movimiento que pueden detectar la presencia de alguien en una habitación (en vez de utilizar coordenadas), pero no pueden distinguir entre personas. Estos sensores de movimiento pueden fallar y producir un falso negativo o un falso positivo (por ejemplo una ventana abierta y una cortina que se mueve) aunque en este escenario supondremos que eso sólo puede ocurrir en contadas ocasiones y por ello tienen una exactitud del 98%.

Si la técnica de localización basada en ZigBee o WIFI no puede determinar con certeza en qué habitación está Lisa (por ejemplo porque las coordenadas relativas están muy próximas a una pared), los sensores de movimiento pueden ayudar a resolver esa incertidumbre. Por otra parte, Lisa ha decidido tener sensores de movimiento sólo en aquellas habitaciones en las que piensa que se puede caer. Hay que destacar que la falta de detección de movimiento por parte de los sensores en ciertas habitaciones puede ser una información útil para decidir en qué habitación está Lisa en algunas situaciones de incertidumbre.

La casa de Lisa tiene 5 habitaciones: recibidor, sala de estar, cocina, baño y dormitorio. Los sensores de movimiento están instalados sólo en tres: cocina, sala de estar y baño. En la siguiente sección se crean paso a paso los modelos de contexto y de aplicación para el caso de estudio descrito.

[1] http://www.modelum.es/MLContext

5 Ejemplo de Modelado del Contexto

5.1 Creación del modelo de contexto

Para ilustrar el uso de MLContext vamos a modelar una pequeña parte del caso de estudio, la información que corresponde a la localización de Lisa en una habitación.

Modelar las Entidades y sus Propiedades. Primero definiremos el usuario Lisa del cual estamos interesados en conocer su posición.

```
entity Lisa context {
      environment {"posicion" source}}
```

Hemos modelado una propiedad "posicion" del tipo *environment* que se refiere a información sobre la localización espacial de las entidades. La palabra clave "source" indica que el valor de esta propiedad se obtiene de una o más fuentes de contexto (en este caso las redes ZigBee o WIFI, que modelaremos más adelante).
A continuación modelaríamos las habitaciones donde se puede localizar a Lisa.

```
entity Recibidor context {}
entity Dormitorio context {}
entity Cocina context {
      computational {"hay_movimiento" source}}
entity Baño context {
      computational {"hay_movimiento" source}}
entity Sala_de_estar context {
      computational {"hay_movimiento" source}}
```

Hemos añadido la propiedad "hay_movimiento" de tipo *Computational*, pero podría definirse también de tipo *Environment*, a la Cocina, el Baño y la Sala_de_estar por tratarse de las habitaciones que tienen sensores de movimiento. El tipo de contexto computacional se refiere a información proporcionada por sistemas hardware o software. La propiedad "hay_movimiento" tendrá un valor de tipo booleano que se obtendrá de una fuente de contexto (el sensor de movimiento que hay en esa habitación).

Especificar las Categorías. Una vez modeladas las entidades, podemos especificar una categoría para cada entidad.

```
categories {
  Usuario:Lisa
  Habitacion:Baño,Dormitorio,Recibidor,Cocina,Sala_de_estar}
```

En este caso se han definido las categorías *Usuario* y *Habitación*. Recordemos que MLContext infiere las propiedades de cada categoría a partir de las propiedades de las entidades que forman parte de ella.

Modelar las Fuentes de Contexto. Para completar el modelo del contexto se especifican las fuentes de contexto, en este caso, la red ZigBee , la red WIFI y los sensores de movimiento.

```
contextSource ZigBee {
      interfaceID : "ZB01"
      methodName : "getZBPosicion" {
          supply : Lisa.posicion
          returnValue : "CoordenadasZB"}  }
```

Se ha especificado que la fuente de contexto ZigBee tiene asociada una interfaz "ZB01".que se refiere al controlador de la fuente. Esta interfaz incluye el método "getZBPosicion()" que se usará para obtener la posición de Lisa en el formato "CoordenadasZB". El middleware deberá incluir una función para obtener a qué habitación de la casa corresponden esas coordenadas. De igual forma se modelaría la red WiFi.

Finalmente habría que especificar una fuente de contexto para cada sensor de movimiento. Por razones de espacio, sólo se muestra cómo se modelaría el sensor de movimiento del baño. Los sensores de movimiento de la cocina y de la sala de estar se modelarían de la misma forma. Todas estas fuentes de contexto devuelven un valor de tipo booleano para indicar si hay o no movimiento en una habitación.

```
contextSource Sensor_baño {
      interfaceID : "BM01"
      methodName : "getMovimiento" {
          supply : Baño.hay_movimiento
          returnValue : "boolean"}}
```

Una vez completado el modelo de contexto se crearía el modelo de aplicación que especificaría los parámetros de calidad para las fuentes de contexto y las situaciones de contexto para nuestro caso de estudio.

5.2 Creación del Modelo de Aplicación

Los elementos del modelo de aplicación harán referencia a los elementos del modelo de contexto.

Especificar los Parámetros de los Sensores. Sólo se considerarán dos parámetros de calidad de las fuentes que pertenecen a la categoría de adquisición de los datos: la exactitud y las unidades en las que suministra la información. Empezaremos por la red ZigBee:

```
provider ZB01 {
      location : "Casa_de_Lisa"
      method : getZBPosicion {
          accuracy : "0.5"
          units : "metros"}  }
```

El identificador "ZB01" relaciona este proveedor de información de contexto con la fuente de contexto ZigBee en el modelo de contexto. Podemos especificar dónde está situado, en este caso se trata de una descripción textual que indica que está en la casa de Lisa, pero como veremos también puede indicarse una entidad. También se ha especificado que los valores obtenidos (usando el método *getZBPosicion*) tienen una exactitud de 0,5 m. El modelado del proveedor para la fuente de contexto WIFI se haría de forma similar, pero con una exactitud de metro y medio.

La especificación de los parámetros de calidad de los sensores de movimiento podría ser la siguiente.

```
provider BM01 {
      location : Baño
      method : getMovimiento {
      accuracy : "98%"}}
```

En este caso, al indicar dónde está situado el sensor, en vez de utilizar una descripción textual, lo hemos situado en la entidad "Baño". Aunque este no es el caso, situar un proveedor en una entidad es muy útil cuando su posición no es fija y puede variar (por ejemplo un teléfono móvil que lleve el usuario o un GPS situado en un automóvil). El valor de "98%" en "accuracy" indica que sólo se produce un error un 2% de las veces en la detección de movimiento. Por razones de espacio no mostraremos cómo se modelan los otros dos sensores de movimiento.

Una vez especificados los parámetros de calidad, el siguiente paso es definir las situaciones de contexto que nos interesan para nuestro escenario.

Modelar Situaciones de Contexto. En este escenario vamos a definir una situación dónde el middleware deba manejar diferentes parámetros de calidad para evaluar si se está produciendo dicha situación. Como ejemplo definiremos una situación para saber si Lisa está en el baño o no.

Dadas las fuentes de contexto disponibles en el escenario, el middleware podría calcular la probabilidad de que Lisa esté en el baño de tres formas distintas: 1) Utilizando una de las redes (ZigBee o WIFI), 2) Utilizando los sensores de movimiento, 3) Utilizando la red y los sensores de movimiento combinados

```
situation Lisa_esta_en_el_baño {
      (#Lisa.posicion==#Baño)
      OR
      (#Baño.hay_movimiento==true AND
      #Cocina.hay_movimiento==false AND
      #Sala_de_estar.hay_movimiento==false)
      OR
      (#Lisa.posicion==#Baño AND
      #Baño.hay_movimiento==true)   }
```

A partir de las situaciones es posible generar de forma automática el código que es capaz de calcular las probabilidades, a partir de la información del modelo de contex-

to, y que se puede utilizar para evaluar la calidad o hacer simulaciones. El lenguaje soporta la composición de situaciones de modo que el ejemplo anterior se podría escribir de la siguiente manera para ganar en claridad, supuesto que se ha definido una situación para cada alternativa:

```
situation Lisa_esta_en_el_baño {
      Red_localiza_a_Lisa_en_el_baño()
      OR Sensor_localiza_a_Lisa_en_el_baño()
      OR Red_y_sensor_localizan_a_Lisa_en_el_Baño ()}
```

Especificar Requisitos de Calidad en las Situaciones de Contexto. Con MLContext es posible también realizar un filtrado de las fuentes que no cumplan unos determinados requisitos de calidad de manera que el middleware no las utilice aunque estén disponibles. Por ejemplo, si quisiéramos descartar cualquier fuente de contexto para determinar la posición de Lisa cuya exactitud fuera peor de 0.6 metros (con lo cual no se consideraría nunca la red WIFI), usaríamos el requisito "accuracy", También podemos especificar que una parte de la definición tiene mayor relevancia, por ejemplo si quisiéramos especificar que preferimos que la aplicación le de preferencia al uso combinado de las dos fuentes de contexto podemos aumentar la relevancia de esa parte de la definición con el parámetro "relevance" (MLContext tiene cinco grados de relevancia predefinidos).

```
situation Lisa_esta_en_el_baño {
      (#Lisa.posicion==#Baño [accuracy < "0.6"])
      OR
      (#Baño.hay_movimiento==true AND
      #Cocina.hay_movimiento==false AND
      #Sala_de_estar.hay_movimiento==false)
      OR
      (#Lisa.posicion==#Baño [relevance == "HIGH"] AND
      #Baño.hay_movimiento==true) }
```

MLContext permite también parametrizar las situaciones, de forma que la definición anterior se podría definir de manera genérica para cualquier entidad del tipo *Usuario*.

5.3 Generación de Código

Como hemos indicado es posible generar código Java que calcule la probabilidad de que ocurra la situación. La transformación modelo a código se realiza utilizando el lenguaje MOFScript, desde el entorno Eclipse. Un extracto del código puede verse en la Figura 2. La clase *Lisa_esta_en_el_baño* implementa el interfaz *IContextQuality* que, a través del método *quality()* nos devuelve su valor. El parámetro *relevance* que especificamos en el modelo modifica el comportamiento del código generado de forma que el valor de probabilidad que devuelve la expresión en la que aparece sea mayor.

```
public class Lina_esta_en_el_baño implements IContextQuality {
     private float CF1;       //Lisa.posicion==Baño
     private float CF2;       //Baño.hay_movimiento==true
     private float CF3;       //Baño.hay_movimiento==true
     private float CF4;       //Cocina.hay_movimiento==false
     private float CF5;       //Sala_de_estar.hay_movimiento==false
     private float CF6;       //Lisa.posicion==Baño

     public Lisa_esta_en_el_baño() {
       ContextFactEvaluator CFE = new ContextFactEvaluator ();
       List<QualityReq> LQR1 = new ArrayList<QualityReq> ();
       LQR1.add(new QualityReq(QualityToolsPackage.QRRelevance,QualityToolsPackage.OPEQUAL,"HIGH");
       CF1=CFE.evaluate("Lisa.posicion","Baño",QualityToolsPackage.OPEQUAL,LQR1);
       CF2=CFE.evaluate("Baño.hay_movimiento","true",QualityToolsPackage.OPEQUAL,NULL);
       CF3=CFE.evaluate("Baño.hay_movimiento","true",QualityToolsPackage.OPEQUAL,NULL);
       CF4=CFE.evaluate("Cocina.hay_movimiento","false",QualityToolsPackage.OPEQUAL,NULL);
       CF5=CFE.evaluate("Sala_de_estar.hay_movimiento","false",QualityToolsPackage.OPEQUAL,NULL);
       List<QualityReq> LQR6 = new ArrayList<QualityReq> ();
        LQR6.add(new QualityReq(QualityToolsPackage.QRAccuracy,QualityToolsPackage.OPLESS,"0.6");
       CF6=CFE.evaluate("Lisa.posicion","Baño",QualityToolsPackage.OPEQUAL,LQR6);
     }

     public float quality() {
       float qualityValue=0.0;
       qualityValue=Math.max(Math.max((CF1*CF2),(CF3*CF4*CF5)),(CF6));
       return qualityValue;
     }
}
```

Fig. 2. Extracto del código Java generado

Además del código anterior, el enfoque propuesto en este artículo nos ha permitido extender la generación de código que presentamos en [1] para el middleware JCAF, incluyendo ahora el manejo de QoC mediante la generación de los métodos que proporciona para ello. También hemos generado la ontología para el manejo de la incertidumbre y las extensiones de calidad que se presenta en [10].

6 Validación de la Propuesta

Para evaluar el modelo hemos seguido el enfoque propuesto en [11], simulando el ejemplo con ayuda del código generado y analizando los resultados cuantitativos. Hemos comparado los resultados con y sin parámetros de calidad, simulando varios escenarios. En el primero consideramos que las dos redes están activas, por lo que el middleware selecciona la red ZigBee por tener mayor exactitud. En el segundo desactivamos la red ZigBee, de manera que el middleware se ve obligado a utilizar la red WIFI. Finalmente, para cada uno de los dos escenarios consideramos también la posibilidad de que haya otra persona en la casa además de Lisa.

Se generaron 10000 posiciones aleatorias de Lisa dentro de la casa. De ellas, 1204 estaban localizadas dentro del baño. Los resultados de la simulación pueden verse en las siguientes tablas. La columna de falsos positivos son los casos en los que el middleware reporta que Lisa está en el baño cuando en realidad está en otro lugar de la casa. La columna de falsos negativos son los casos en los que Lisa está en el baño pero el middleware no lo detecta. Finalmente la columna de aciertos son los casos en los que Lisa está en el baño y el middleware lo detecta correctamente.

Tabla 1. Resultados de la simulación utilizando la red ZigBee y los sensores.

Red ZigBee	Lisa sola en la casa			Hay otra persona más		
	Falsos positivos	Falsos negativos	Aciertos	Falsos positivos	Falsos negativos	Aciertos
sin QoC	138	7	1197	393	71	1133
con QoC	136	5	1199	388	33	1171

En la Tabla 1 están los resultados considerando que el middleware utiliza la red ZigBee (mayor exactitud). Observamos que la forma en la que hemos definido la situación, combinando la información de los sensores con la de la red ZigBee es suficiente para detectar casi todos los casos en los que Lisa está en el baño y está sola en la casa, incluso sin utilizar los parámetros ni requisitos de calidad del modelo. Al utilizar la información de calidad contenida en el modelo se reduce un poco la cantidad de falsos positivos y la de falsos negativos.

Si consideramos la situación con otra persona en la casa, la información de los sensores de movimiento ya no es tan útil. En este caso el número de aciertos es menor y se produce un aumento considerable de falsos positivos y también se introducen falsos negativos. Vemos que al utilizar los parámetros de calidad mejora un poco (3,35%) la tasa de aciertos y se reduce un poco (1,27%) el número de falsos positivos y bastante (53,52%) el de falsos negativos.

En la Tabla 2, podemos ver los resultados obtenidos para las mismas 10000 posiciones pero utilizando la red WIFI. Si comparamos los resultados con los de la primera tabla, observamos que aumenta el número de falsos positivos, debido a que la exactitud de la red WIFI es peor, aunque aún es capaz de detectar bastantes casos en los que Lisa está en el baño cuando está sola en la casa. Cuando hay otra persona en la casa los resultados son considerablemente peores, aunque el uso de parámetros de calidad reduce mucho (62,67%) el número de falsos negativos y mejora la tasa de aciertos (13,78%).

Tabla 2. Resultados de la simulación utilizando la red WIFI y los sensores.

Red WIFI	Lisa sola en la casa			Hay otra persona más		
	Falsos positivos	Falsos negativos	Aciertos	Falsos positivos	Falsos negativos	Aciertos
sin QoC	305	19	1185	541	217	987
con QoC	222	9	1195	500	81	1123

De las dos tablas podemos sacar como conclusión que el uso de la información de los parámetros y requisitos de calidad contenidos en el modelo reducen un poco la cantidad de falsos positivos y mejoran también un poco la tasa de aciertos, aunque la principal mejora se da en el número de falsos negativos, siendo esta mejora mayor cuanto peor es la exactitud de la red.

7 Trabajo Relacionado

Hasta ahora se ha prestado muy poca atención a la aplicación de técnicas MDE en el desarrollo de aplicaciones context-aware que tengan en cuenta la calidad de contexto, y podemos destacar los trabajos presentados en [15] y [8]. COSMOS [15] es un manejador de contexto con soporte para la QoC que aplica MDE para generar los artefactos que manejan el contexto a partir de modelos. Al igual que nuestro enfoque se separa entre el modelo de contexto y el modelo de aplicación. Una diferencia significativa con MLContext es el hecho que COSMOS sea una solución orientada a un manejador de contexto concreta, mientras MLContext es genérico. Además no se considera la definición de contexto abstracto.

A partir de la experiencia adquirida con COSMOS, se ha publicado recientemente el metamodelo QoCIM cuyo objetivo es definir un metamodelo para QoC que sea genérico (independiente de cualquier manejador de contexto), con una expresividad apropiada y cuyos modelos sean ejecutables. Los autores se han inspirado en un metamodelo de métricas software para ofrecer una solución que permita definir cualquier tipo de parámetro de calidad. De momento, QoCIM es un trabajo inicial y todavía no soporta la definición de requisitos y niveles de calidad y no se disponen de librerías de modelos que definan parámetros de calidad comúnmente usados. MLContext usa un conjunto de parámetros de calidad que son los que normalmente se manejan y cuya semántica es bien conocida y debe ser implementada en los middleware y cabe destacar que no está ligado a un manejador concreto.

Un enfoque de QoC basado en UML se propone en [11]. Al igual que el nuestro, permite modelar la calidad tanto a nivel de los sensores como de situaciones de contexto. Cabe destacar que a pesar de ser un trabajo desarrollado cuando las técnicas de MDE habían madurado, no se ofrece una solución MDE articulada en torno a un metamodelo, sino que se usan simplemente modelos UML. El trabajo plantea que los modelos de calidad se pueden integrar de forma transparente en formalismos como el algoritmo de voting o la lógica difusa, pero no en otros como las redes bayesianas. Se realiza una simulación sobre un escenario para evaluar cómo el uso de parámetros de calidad afecta a la detección de situaciones de contexto. Nuestra evaluación se ha basado en una simulación similar. Los resultados de su evaluación no son concluyentes debido, según los autores, al mecanismo de razonamiento aplicado (algoritmo de voting) y no al modelo propuesto. A diferencia de nuestra propuesta, ellos no tienen en cuenta los requisitos de calidad del usuario.

8 Conclusiones

Se ha presentado un método basado en el DSL MLContext para el modelado del contexto con QoC en sistemas context-aware. Hemos ilustrado la aplicación del método con un caso de estudio de AAL, habiendo mostrado cómo especificar parámetros y requisitos de calidad, y hemos evaluado por medio de una simulación la influencia de los parámetros de calidad utilizando código Java generado para un requisito de calidad. En los resultados de la simulación se observa que la mejora es

mayor en situaciones adversas (cuando hay otra persona en la casa) y cuando las fuentes de contexto tienen unos parámetros de calidad bajos (mala precisión). A partir de modelos MLContext es posible definir artefactos relacionados con la calidad como código Java que permite calcular la probabilidad de que una situación de contexto se esté produciendo en un instante determinado o instancias de ontologías para QoC. Como trabajo futuro estamos colaborando con el grupo DistriNet de la universidad KU Leuven para la elaboración de un framework de manejo de contexto en el que los modelos de las situaciones de contexto se utilizarían para generar funciones JavaScript que serían inyectadas dinámicamente.

Agradecimientos. Este trabajo ha sido subvencionado con el proyecto SociAAL-UMU (TIN2011-28335-C02-02 Ministerio de Ciencia e Innovación), y la ayuda 15389/PI/10 (Fundación Seneca, Agencia de Ciencia y Tecnología Región de Murcia)

Referencias

1. Hoyos, J.R., García-Molina, J.J., Botía, J.A.: A domain-specific language for context modeling in context-aware systems. Journal Syst. And Softw., 86 (11), pp. 2890-2905, (2013)
2. Hoyos, J.R., Preuveneers, D., García-Molina, J.J., Berbers, Y.: A DSL for Context Quality Modeling. Advances in Intelligent and Soft Computing, vol 92, pp 41-49, (2011)
3. Ayed, D., Delanote, D., Berbers, Y.: MDD approach for the development of context-aware applications. In CONTEXT 2007. Vol. 4635. LNAI. , pp. 15–28 (2007)
4. Sindico, A., Grassi, V.: Model driven development of context aware software systems. In Int. Workshop on Context-Oriented Programming, Genova, Italy. ACM, pp. 1–5, (2009)
5. Buchholz, T., Küpper, A., Schiffers, M.: Quality of context information: What it is and why we need it. In: Proc. of the 10th Int. Workshop of the HP OpenView University Association (HPOVUA). vol 2003., Geneva, Switzerland, (2003)
6. Wand, Y. and Wang, R.Y. Anchoring Data Quality Dimensions in Ontological Foundations. Communications of the ACM. vol 39. no. 11, (1996)
7. Wang, R. and Strong, D. “Beyond Accuracy: what Data Quality Means to Data Consumers”, Journal of Management Information Systems, vol. 12, no. 4, pp. 5-34, (1996)
8. Marie, P., Desprats, T., Chabridon, S., Sibilla, M..: QoCIM: A Meta-model for Quality of Context. Modeling and Using Context. LNCS vol. 8175, pp 302-315 (2013)
9. AAL Open Association (AALOA), http://www.aaloa.org/
10. Preuveneers,D., Berbers, Y.: Quality Extensions and Uncertainty Handling for Context Ontologies. In proceedings of the (C&O-2006). Riva del Garda, Italy,(2006).
11. McKeever, S.,Ye, J.,Coyle, L.,Dobson, S.: A Context Quality Model to Support Transparent Reasoning with Uncertain Context. Quality of Context. LNCS 5786,pp 65-75, (2009)
12. Krause, M., Hochstatter, I.: Challenges in Modelling and Using Quality of Context (QoC). Mobility Aware Technologies and Applications. LNCS 3744, pp 324-333, (2005)
13. Sheng, Q.Z., Benatallah, B.: ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In: International Conference on Mobile Business (ICMB 2005), Sydney, Australia , pp.206–212, (2005)
14. Baldauf, M.: A survey on context-aware systems. Int. J. Ad Hoc and Ubiquitous Computing, Vol. 2, No. 4, (2007)
15. Chabridon,S.Conan,D.Abid,Z.,Taconet,C.:Building ubiquitous QoC-aware applications through model-driven software engineering.Sci Comput.Program.78(10):1912-1929 2013

Una solución MDE para crear aplicaciones basadas en mensajes instantaneos a través de Twitter

Ángel Mora Segura, Juan de Lara y Jesús Sánchez Cuadrado

Grupo de Modelado e Ingeniería del Software (http://www.miso.es)
Departamento de Informática
Universidad Autónoma de Madrid
{Angel.MoraS, Juan.deLara, Jesus.Sanchez.Cuadrado}@uam.es

Resumen. Las redes sociales se utilizan para todo tipo de propósitos, en cualquier momento y lugar, gracias al auge de los dispositivos móviles. La capacidad de envío de mensajes (públicos o privados) de estos servicios se usa hoy día, de manera habitual, para segmentar al conjunto de usuarios según preferencias, opiniones y eventos. Esto no pasa desapercibido para empresas o servicios que analizan datos mediante la nueva figura del llamado *social media manager*.
Como ejemplo de servicio, debido a su carácter eminentemente público, los *tweets* de Twitter pueden ser vistos como un mecanismo de diálogo entre usuarios y aplicaciones informáticas. Resaltando esta funcionalidad, proponemos el concepto de *aplicación basada en mensajes instantáneos*, una aplicación que usa los mensajes emitidos por los usuarios como un medio para obtener comandos de entrada y emitir salidas. Para ello, usaremos las ventajas del Desarrollo de Software Dirigido por Modelos para la construcción de un prototipo (llamado *Twiagle*) que permita de una forma clara y concisa, mediante lenguajes de dominio específico: (i) detectar patrones en los mensajes de los usuarios, y (ii) construir aplicaciones simples mediante acciones, tales como consultas a los mensajes seleccionados y síntesis de mensajes.

Palabras clave: Desarrollo de Software Dirigido por Modelos, Ingeniería basada en la Web, Lenguajes de Dominio Específico, Interfaces de Usuario basadas en Mensajes, Twitter.

1 Introducción

Gracias a la proliferación de los llamados dispositivos móviles, el uso de las redes sociales se ha disparado en los últimos años. Servicios como Twitter[1] o Google+[2] se utilizan para conectar a personas según sus gustos, aficiones o en torno a eventos concretos. Esta particularidad no sólo ha sido aprovechada por

[1] http://www.twitter.com
[2] https://plus.google.com/

los usuarios, sino tambien por empresas, que han visto en las redes sociales una oportunidad para acercarse a sus clientes [20].

En este sentido, hemos observado la necesidad de automatizar ciertas actividades sociales, sobre todo en plataformas que permitan el acceso a los mensajes emitidos de una forma pública. Por otro lado, los usuarios de estos servicios, con conocimientos o no de programación, pueden querer definir aplicaciones colaborativas sencillas. Por ejemplo, las empresas frecuentemente interaccionan con otros usuarios a través de la nueva figura denominada *social media manager* para gestionar la relación con sus clientes. Aunque este rol esté tomando cada vez más importancia, la mayoría de tareas asociadas a esta relación son llevadas a cabo de manera manual o creando aplicaciones destinadas específicamente para una tarea en concreto.

Nuestra teoría se basa en la idea de que las redes sociales basadas en sistemas de mensajería instantanea se pueden convertir en la interfaz de usuario necesaria para aplicaciones informáticas concretas. Este trabajo no pretende sustituir a otras herramientas basadas en un acceso multicanal a la información, mediante la definición de su arquitectura al completo [6] o el modelado de un flujo estándar de consultas a mensajes almacenados en redes sociales [5], sino que pretende dar una solución factible para usuarios no expertos que quieran desarrollar aplicaciones de una forma rápida y adaptada a un escenario concreto. Sin embargo, se han observado los siguientes hechos que ilustran las ventajas que este tipo de aplicaciones pueden presentar en distintos escenarios:

- los sistemas de redes sociales ofrecen una interfaz robusta, con la que los usuarios ya están sobradamente familiarizados, y que puede ser reutilizada para evitar la implementación de soluciones más específicas,
- estos sistemas están ya instalados en un gran número de dispositivos y además están diseñados para soportar una carga de usuarios y mensajes muy alta [2, 3], y
- las aplicaciones concretas pueden heredar como suyas las ventajas que estos sistemas presentan.

En adelante, usaremos Twitter como ejemplo a través de lo que hemos llamado como aplicaciones basadas en *tweets* e interfaz de usuario basada en *tweets*. Proponemos una aproximación basada en Desarrollo de Software Dirigido por Modelos (DSDM, o MDE las siglas en inglés) para automatizar y sistematizar su desarrollo. Esta solución incluye: (a) un Lenguaje de Dominio Específico (DSL) para expresar patrones conectado a Wordnet [18], (b) un DSL para describir consultas y acciones sobre los mensajes seleccionados y (c) un prototipo, llamado *Twiagle*, construido dentro del entorno Eclipse para modelar y desarrollar las aplicaciones.

En la siguiente sección describiremos tres escenarios lo suficientemente relevantes para que se observe la variedad de usos que permite esta solución. En el primero de ellos, Twitter será usado como interfaz de usuario que necesita estar conectado a un sistema de información ya existente. En el segundo de ellos, se describirá cómo crear aplicaciones simples y pequeñas que puedan ser diseñadas

por usuarios finales no expertos. En el último, y bajo una situación de emergencia o de un evento inesperado, se mostrará la construcción rápida de una aplicación para coordinar a un gran número de personas. Todas estas aplicaciones necesitan soportar un gran número de usuarios, ser altamente accesibles y beneficiarse de metainformación almacenada en los *tweets*, como es el caso de la geoposición.

Este artículo extiende nuestro trabajo previo en [23], donde se presentaba *Twiagle* brevemente desde el punto de vista de la herramienta en sí. En este artículo desarrollamos los conceptos principales, y explicamos los DSLs elaborados con detalle.

El resto de las secciones están organizadas de la siguiente forma: presentación de los lenguajes diseñados en las secciones 4 y 5, un prototipo de la herramienta que soporta el diseño de las aplicaciones en la sección 6, y por último, algunas conclusiones obtenidas en la sección 8.

2 Motivación

Si bien el concepto de aplicación basada en mensajes instantáneos se puede exportar a otras redes sociales, debido a su popularidad, hemos escogido Twitter. Twitter es una red social basada en el *microblogging*, que permite una comunicación directa entre usuarios mediante mensajes cortos de 140 caracteres, llamados *tweets*. Un tweet puede contener referencias a otros usuarios ('@'), *hashtags* (palabras predefinidas por los propios usuarios precedidas por '#'), enlaces, e incluyen información implícita como su fecha de publicación o su geolocalización. Los tweets son públicos por defecto, cosa que no ocurre en el resto de redes sociales del mismo tipo, y por tanto, cualquier persona puede verlos, sea usuario de Twitter o no. Hay también mensajes privados, que son enviados a usuarios específicos. En Twitter, los usuarios pueden seguir a otros usuarios, recibiendo el que sigue el nombre de *follower*, lo que a diferencia de otras redes sociales no implica que el usuario *seguido* entable relación alguna con el primero.

A continuación describimos algunos escenarios donde las aplicaciones basadas en mesanjes instantáneos pueden resultar de utilidad.

1. **Interfaces basadas en Tweets para Sistemas de Información**. En este escenario, una compañia decide usar Twitter para mejorar el acceso a su sistema de información. La ventaja es que los usuarios no necesitan instalar software nuevo para interaccionar con el sistema, ya que pueden usar su cuenta de Twitter desde móviles, PCs, o portátiles. Además, se puede extraer información de la localización y usar la estructura social de Twitter para enviar mensajes *broadcast*.
   Como ejemplo, la Figura 1 muestra la integración de una interfaz basada en tweets con el sistema de información de un aeropuerto. El objetivo es informar a los viajeros del estado de sus vuelos, entre otros servicios. En el paso 1, un viajero envía un tweet solicitando información sobre un vuelo. El tweet menciona la cuenta del aeropuerto, por ejemplo *@aena*. En el paso 2, la interfaz recibe el tweet, comparándolo con una colección de patrones

que identifican peticiones. Si se encuentra una coincidencia, la petición se redirige al sistema de información (paso 3), que busca el vuelo en la base de datos. Finalmente (paso 4), la interfaz genera un mensaje privado para el usuario. En el ejemplo, el mensaje inicial es público, pero podría haber sido privado.

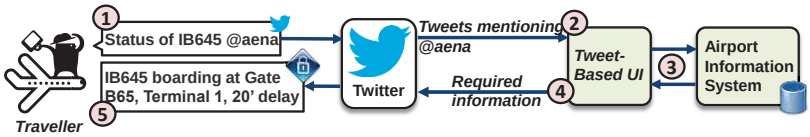


Fig. 1: Una interfaz basada en tweets para el sistema de información de un aeropuerto.

2. **Aplicaciones colaborativas *ad-hoc* basadas en Tweets**. En algunos casos, los usuarios querrían usar Twitter para la recogida de información y su posterior procesamiento automatizado. No obstante, pueden carecer de las habilidades técnicas para desarrollar una aplicación de este tipo. Esta situación puede incluir: votaciones, organización de eventos deportivos, juegos educativos en movilidad [17], entre muchos otros. Por ejemplo, en el caso de los juegos educativos, el profesor podría diseñar un cuestionario, que los estudiantes tendrían que responder durante una visita a un museo, pudiendo colaborando a través de Twitter. A lo largo del artículo, usaremos como ejemplo una aplicación simple para la realización de votaciones.
3. **Aplicaciones masivamente paralelas para situaciones inesperadas** Las tecnologías de la información desempeñan un papel crucial para la respuesta ante emergencias [9]. Por ejemplo, Twitter fue un medio de distribución de información del gobierno japonés por el desastre de la central nuclear de Fukushima [16]. En este tipo de aplicaciones, la información de geoposición incluida en los tweets puede ser valiosa. Otro ejemplo sería la construcción de un *front-end* para reasignar vuelos en caso de una huelga. Esto es útil por dos razones: el sitio web de la compañia podría no soportar la carga debido al acceso de un gran número de usuarios a la vez, y los usuarios no necesitan instalar una aplicación nueva en sus móviles.

El objetivo de nuestro trabajo es el desarrollo de una aproximación que permita una construcción fácil y rápida de este tipo de aplicaciones, incluso por usuarios sin grandes conocimientos en informática.

3 Arquitectura de la solución

El esquema de trabajo de nuestra solución para aplicaciones basadas en tweets se muestra en la Figura 2, donde los números ilustran una interacción típica.

En primer lugar (1), los usuarios envían tweets o mensajes privados a través de Twitter. La información relevante en los tweets ha de ser entonces extraída

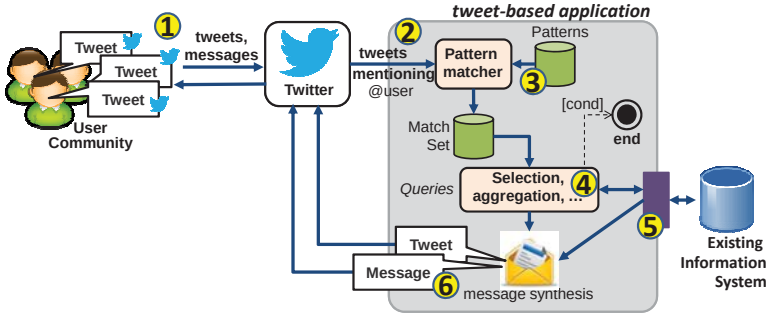


Fig. 2: Esquema de la solución.

por la aplicación. Nuestra solución se basa en la definición de patrones, que se esperan encontrar en los tweets. Para este prototipo, hemos limitado la recepción de tweets a aquellos que mencionan al usuario que administra la aplicación o que le envían un mensaje directo, pero sería también ampliable a otro tipo de criterio.

Los patrones (3) los definiría un experto en *social media*. Un uso típico, incluiría diferentes consultas, seleccionando los conceptos relevantes a encontrar en los tweets o calculando diferentes valores agregados a partir de ellos (4). Además, la información puede ser obtenida o enviada a sistemas de información existentes (5). La información extraída de las consultas o provista por los sistemas de información se puede usae para sintetizar tweets o mensajes privados, que serán enviados a los usuarios (6). Finalmente, pueden definirse condiciones que establezcan el fin de la ejecución de la aplicación.

Nótese que, a diferencia del enfoque propuesto en [5], esta aplicación no se limita a filtrar un listado de tweets según un criterio de búsqueda estático, sino que es capaz de hacer búsquedas basadas en relaciones complejas entre distintos conceptos, agregar la información, sintetizar mensajes y presentar los resultados en tiempo real, permitiendo al usuario elegir qué información quiere extraer de la salida propuesta.

Para facilitar la implantación de un sistema de estas características, proponemos una solución basada en DSDM, que consta de dos DSLs como muestra la Figura 3. El primer DSL (llamado *Twittern*) sirve para definir patrones y los conceptos que lo forman.

El segundo DSL, llamado *Twition* permite definir consultas sobre tweets que coinciden con algún patrón, utilizando una sintaxis de estilo similar a SQL. Las siguientes dos secciones describen los dos DSLs en detalle.

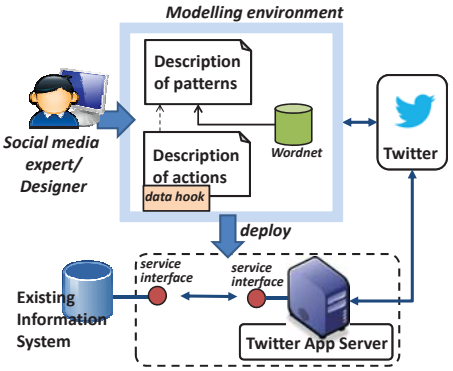


Fig. 3: Visión general del enfoque.

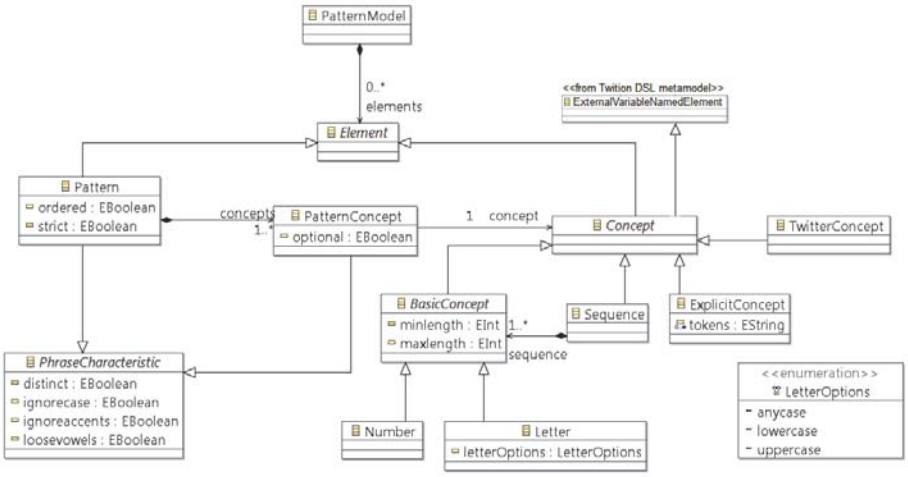


Fig. 4: Fragmento simplificado del meta-modelo de Twittern.

4 Twittern: un lenguaje para definir patrones

Hemos desarrollado un DSL, llamado *Twittern*, para facilitar la definición de patrones a ser identificados en tweets. La Figura 4 muestra un fragmento del metamodelo de dicho DSL. Un patrón está formado por conceptos, y en su forma más simple, un concepto (ExplicitConcept) es un conjunto de palabras (tokens). Este conjunto puede ser o bien definido explícitamente por el diseñador o bien automáticamente tomado del conjunto de sinónimos que proponga WordNet [18]. También hemos definido conceptos especiales para los números y las letras (representados en las subclases de BasicConcept).

El DSL también incluye conceptos específicos de Twitter (como subclases de TwitterConcept, aquí omitidas), como nombres de usuarios, URLs (especialmente imágenes), y para definir colecciones de hashtags interesantes. Como trataremos en la siguiente sección, los metadatos de los tweets, como el autor, la fecha o la geoposición se pueden obtener sin declararse explícitamente en los patrones.

La clase Concept hereda de ExternalVariableNamedElement, lo que permite que los conceptos coincidentes sean enviados como datos externos (ver siguiente sección), y que sean referenciados en las consultas. Como se puede apreciar, hemos separado la definición de los conceptos de su uso en los patrones, lo que permite su reutilización en diferentes contextos. Una clase intermedia PatternConcept permite la configuración de cómo se debe usar el concepto, especificando por ejemplo si es opcional en el patrón. Los patrones también indican si los conceptos tienen que aparecer en un orden específico o permitir que aparezcan otras palabras entre los conceptos. Asimismo, es posible definir que alguno de los conceptos puede no aparecer en un patrón, y si los conceptos deben identificarse ignorando mayúsculas, ignorando acentos, o permitiendo que falten vocales para dar una mayor libertad a los usuarios.

En resumen, el DSL pretende ser más simple de las expresiones regulares de cara al usuario final, estar más adaptado a mensajes instantáneos y Twitter, además de apoyarse en la potencia que ofrece Wordnet. *Twittern* tiene una sintaxis textual muy sencilla. Como ejemplo, la Figura 5 muestra algunos patrones y conceptos para una aplicación sencilla, para la realización de una votación.

```
1  pattern voteY [ordered loose−vowels ignore−case] = {question?, yes}
2  pattern voteN [ordered loose−vowels ignore−case] = {question?, no}
3
4  concept question = {question, inquiry, enquiry, query, interrogation}
5  concept yes = {yes, affirmative, 1, true, y}
6  concept no = {no, negative, 0, false, n}
```

Fig. 5: Patrones para el ejemplo de votación.

El objetivo de la aplicación es detectar votaciones positivas o negativas a preguntas sencillas. Los patrones que las detectan se muestran en las lineas 1–2. Se etiquetan como ordered, de tal manera que los conceptos deben aparecen en orden, pero no son strict, y de esta manera otros elementos pueden aparecer en en tweet, intercalados con los conceptos del patrón. Sólo los conceptos yes y no son obligatorios, mientras que el concepto question es optativo (indicado por ?). Para todos los conceptos, se ignora si están en mayúsculas o minúsculas (ignore-case), y admitimos variaciones con posibles vocales que falten (loose-vowels). Los patrones que usan los conceptos se definen en las líneas 4–6. Las palabras que forman el concepto *question* se han sacado de los sinónimos de Wordnet. De esta manera, *voteY* podrá detectar tweets como *"My vote 4 the qstn is YES!"*. En este caso, el valor real de question es qstn mientras que el valor de yes es YES.

5 Twition: un lenguaje para definir acciones

En nuestra propuesta, las acciones necesarias para la aplicación se expresan mediante otro DSL, llamado *Twition*. La Figura 6 muestra un fragmento de este metamodelo (muchas clases concretas han sido omitidas, dejando sólo las clases base abstractas de las principales jerarquías). *Twition* usa *Twittern* para hacer consultas sobre los conceptos encontrados en los tweets que cumplen un determinado patrón, y además soporta otras acciones que describimos a continuación.

– **Consultas**. Utilizando una sintaxis del estilo de SQL, pueden referirse a un conjunto de tweets de los que cumplen un patrón como si formasen una tabla SQL, y a los conceptos del patrón como si fuesen columnas SQL. También pueden consultarse los metadatos de los tweets, utilizando palabras reservadas como user, geo, y date. Hay tres tipos de consultas: Select, Adding y Metadata. Select (clase PatternSelect) selecciona una serie de conceptos del conjunto de tweets que cumplen un patrón; Adding (clase AddingSelect) puede realizar una serie de operaciones aritméticas (*maximum*, *minimum*, *average*

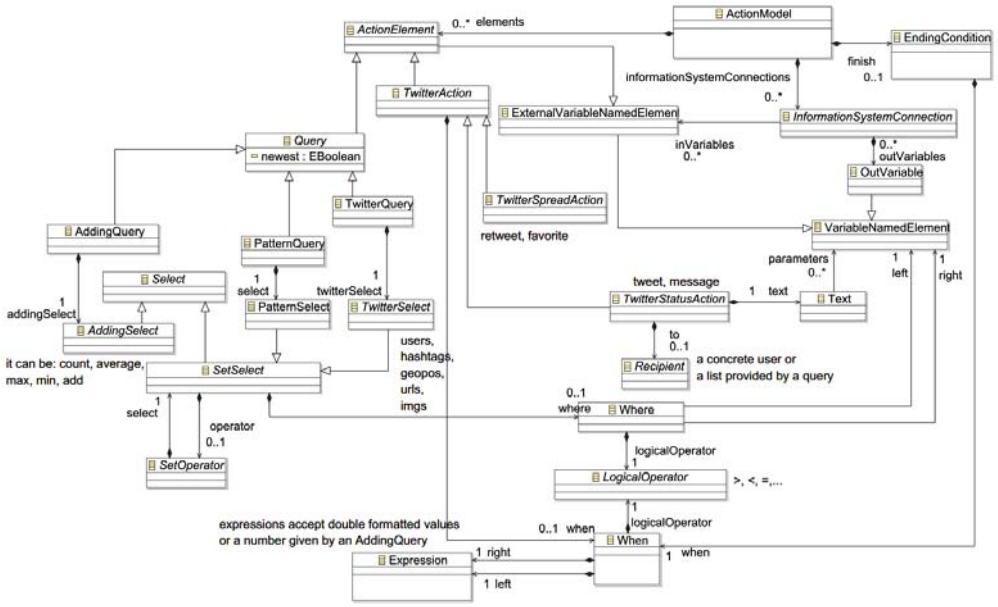


Fig. 6: Una simplificación del meta-modelo de Twition.

y *count*) sobre conjuntos de resultados; y Metadata (clase TwitterSelect) es capaz de obtener un conjunto de resultados compuesto de metadatos. Toda consulta tiene un nombre para que sus resultados puedan ser reutilizados en otras acciones. Todo tipo de consulta, excepto las de tipo Adding, devuelve un conjunto de resultados, y actualmente ofrecemos tres tipos de operaciones con ellos (las representadas como subclases de SetOperator): *union*, *intersection* y *substraction*.

Por ejemplo, la Figura 7 muestra dos consultas de tipo Adding (líneas 2–3) y una consulta de metadatos (línea 6). Las dos primeras consultas cuentan los tweets detectados por los patrones voteY o voteN, contando aquellos tweets que contienen el token yes en cualquiera de sus formas. La última consulta obtiene los usuarios que enviaron un tweet que fue detectado por los patrones, y utiliza el operador union.

Mientras que las consultas tienen una sintaxis similar a la de SQL, los datos de entrada son tomados dinámicamente del sistema de microblogging, y por tanto, del mismo modo que en los *data stream management systems* [4], podríamos usar ventanas temporales [12]. Actualmente, facilitamos dos tipos de ventanas temporales: una considerando todos los datos (lineas 2 y 3) y otra con el último tweet (linea 6, @newest).

– **Confeccionar y enviar mensajes**. Una vez los datos de las consultas están disponibles, se puede confeccionar y enviar mensajes a un grupo de usuarios (TwitterStatusAction). Estos mensajes pueden ser públicos (comando tweet) o privados (comando message). Además, los mensajes recibidos se pueden

```
1  //counting ----------------------------------------
2  query countY : count yes from voteY;
3  query countN : count no from voteN;
4
5  //thanks message----------------------------------------
6  @newest query users_voting : users from voteY union users from voteN;
7  message thanku ("Thanks for your vote.") to users_voting;
8
9  tweet presults ("Partial results: (%s) yes, (%s) no", countY, countN) when (countY + countN) = 15;
10 tweet results ("Results: (%s) yes, (%s) no", countY, countN) when (countY + countN) = 30;
11 finish end: when (countY + countN) = 30;
```

Fig. 7: Acciones para el ejemplo de la votación.

volver a enviar a otros usuarios, que en el caso concreto de Twitter, se haría con los comandos retweet y favorite. Se pueden enviar mensajes públicos a un conjunto de usuarios obtenidos mediante una consulta, además de configurar cuando se quiere enviar ese mensaje (clase When).
Por ejemplo, en la Figura 7 se envía un mensaje privado (linea 7) a los usuarios que han votado. El usuario se obtiene de la consulta users_voting, que obtiene el usuario del último voto positivo o negativo. Las líneas 9 y 10 muestran la síntesis de dos tweets. De manera similar a la función printf de C, el tweet se compone insertando datos (countY, countN) en un string, en los lugares indicados usando '%s' (independientemente del tipo de la variable). Ambos tweets tienen condiciones de disparo, de tal manera que se mandan cuando el número de votos alcanza 15 y 30 respectivamente. También se pueden generar tweets que mencionen usuarios.

- **Fin de la ejecución**. Existe un mecanismo para establecer una condición que determine cuándo ha de producirse el final de la ejecución. La condición puede depender de diferentes factores como el número de tweets recibidos que se ajustan a cierto patrón, o el tiempo de ejecución. Un ejemplo de ello es lo que se muestra en la línea 11 de la Figura 7.
- **Obtener datos externos**. Se ha habilitado un mecanismo (clase InformationSystemConnection) que permita en un futuro, facilitar la conexión de los mensajes recibidos con un sistema de información existente, de forma que se solucione el problema mostrado en el apartado 1 de la sección 2. Este mecanismo está pensado para que el usuario pueda declarar datos que se han de enviar al sistema de información, o bien ser recibidos de una forma asíncrona. Lógicamente, esta funcionalidad estaría pensada para usuarios expertos o con conocimientos previos de programación, ya que, aunque nuestra herramienta generaría las clases básicas para la integración con el sistema de información, estas clases deberían ser rellenadas manualmente.

En nuestro lenguaje, cada acción tiene un nombre, para que otras acciones puedan referirse a los datos que éstas producen. El tipo de datos es inferido por reglas sencillas: los datos que produce un tweet son considerados String, y las consultas de tipo Adding devuelven datos enteros (ver por ejemplo, lineas 9 y 10 de la Figura 7).

El modelo de ejecución de Twition está basado en el flujo de datos, que es el modelo de ejecución recomendado para aplicaciones reactivas, dirigidas por eventos y escalables [21]. De este modo, una acción se ejecuta tan pronto como los datos requeridos están disponibles, a menos que contengan un *disparador* (trigger) específico; en ese caso, se ejecutará cuando los datos estén disponibles *y* cuando la condición del disparador sea cierta.

6 Prototipo

Para dar soporte a ambos DSLs, hemos desarrollado un prototipo de entorno de modelado, en un plugin de Eclipse. Este entorno permite describir patrones y consultas sobre ellos, y hacer tests sobre posibles resultados. Se puede obtener más información sobre la herramienta y un video que la muestra en acción en `http://www.miso.es/tools/twiagle.html`.

La herramienta, llamada Twiagle, soporta un método de desarrollo ágil que permite al usuario testear los diferentes patrones contra tweets recibidos en tiempo real (Figura 8, 1). La herramienta se comunica con Twitter usando la librería Java Twitter4J[3]. Los conceptos pueden ser definidos en tiempo real a través de WordNet, mediante un asistente contextual que permite obtener sinónimos sobre las palabras a buscar, además de ayudar en la escritura de los comandos del lenguaje. La herramienta está diseñada para su uso por usuarios no expertos. Por ello, se han incluido en el asistente algunos ejemplos relevantes de aplicaciones típicas (Figura 8, 2). Además, durante la ejecución se muestra una vista con los tweets que han ido encajando con nuestros patrones y los resultados obtenidos hasta el momento para las acciones definidas (etiqueta 3). Por último, se muestra (etiqueta 4) una sección de la interfaz web de Twitter, donde se puede comprobar los tweets y los mensajes que se han emitido, así como las acciones que se han llevado a cabo.

7 Trabajo relacionado

Cada vez es más frecuente que las aplicaciones web se extiendan con funciones que les permitan interactuar con las redes sociales. Por ejemplo, mostrar todos los tweets de un determinado usuario o hashtag. Para eso, [5] propone una extensión de WebML para incorporar primitivas sociales, como el login o la búsqueda, y acciones específicas, como por ejemplo publicar en un muro de Facebook o un tweet. En nuestro caso dotamos a un usuario inexperto en el desarrollo web y en el modelado de aplicaciones, de las herramientas necesarias para hacer búsquedas complejas sobre una red social y recoger de forma adecuada los resultados de éstas sin que tengan que recurrir para ello al desarrollo de un sistema web específico.

Relacionado con lo anterior, [10] es un método para construir simples recetas para automatizar tareas desde distintos *canales* (como Facebook o Twitter). Un ejemplo de esto sería "tweetear todas mis actualizaciones de estado de

[3] `http://twitter4j.org`

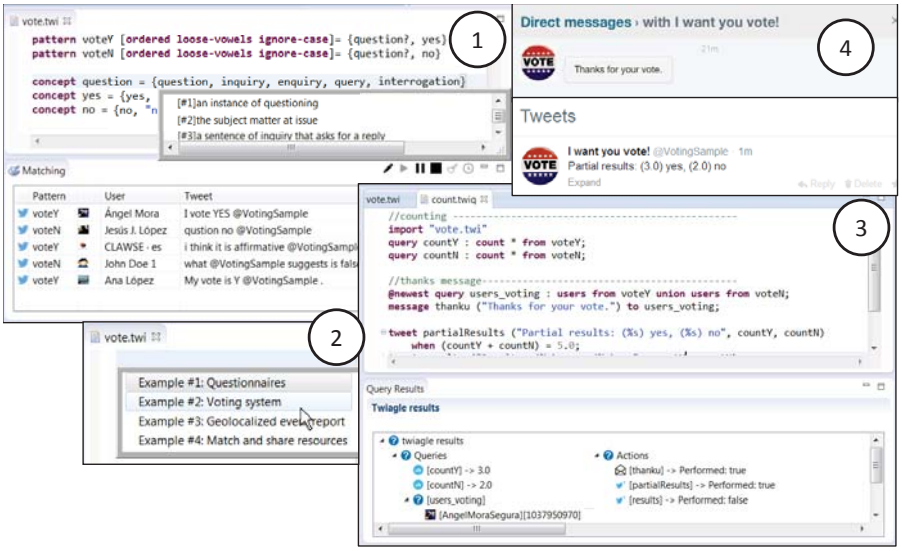


Fig. 8: La herramienta en acción.

Facebook". En Twiagle, permitimos definir tareas más complejas, y facilitamos mecanismos de conexión con otras fuentes de datos, permitiendo así la migración de sistemas de información existentes a aplicaciones sociales.

Otros trabajos utilizan Twitter para crear nuevas aplicaciones desarrolladas *ad-hoc*, sin un framework de apoyo como *Twiagle*. Por ejemplo, en [22], se usan tweets para detectar terremotos en Japón. Su método rastrea una gran cantidad de tweets, discriminándolos dependiendo de si indican la ocurrencia o no de un terremoto. La clasificación se lleva a cabo mediante una máquina de vectores de soporte previamente entrenada. Cada tweet tiene un tiempo y una geoposición asociados, y de ese modo el sistema es capaz de localizar el terremoto y reportarlo a una serie de usuarios. Esta aplicación fue desarrollada de manera específica para su propósito, y creemos que es justo este tipo de soluciones las que podrían beneficiarse enormemente de un entorno como *Twiagle*.

Asimismo, numerosos trabajos se centran en el estudio de la estructura de red de Twitter [13, 26], otros en su uso como un medio social de comunicación. Un ejemplo de esto último es [25], que plantea un mecanismo para detectar mensajes en situaciones de emergencia, usando para ello una combinación de propiedades lingüisticas anotadas manualmente y extraídas automáticamente . En [16], se desarrolla un análisis de contenido de tweets y del comportamiento de los retweets durante la catástrofe nuclear de Fukushima. Otros trabajos [7] tratan de analizar información inherente a los propios tweets, como la geoposición para, por ejemplo investigar patrones de concentraciones de personas. En nuestro caso, nuestro objetivo es detectar patrones sencillos de texto, para lo que por el momento no empleamos técnicas sofisticadas de procesamiento de lenguaje

natural. No obsatante, el lenguaje se apoya en Wordnet, para la extracción de sinónimos, así como de opciones para considerar abreviaturas de palabras (que eliminen alguna vocal), y además se permite la obtención de metainformación básica dentro de los tweets.

Los tweets contienen información desestructurada, y, para hacerla procesable, algunos trabajos [24, 8] proponen la incorporación de información estructurada a los propios tweets, mientras que otros utilizan técnicas de procesamiento de lenguaje natural (NLP). En el primer grupo, en [24], los autores proponen un lenguaje de flujo de trabajo sencillo llamado *tweetsflows*, que introduce algunos conceptos de SOA en Twitter, incluyendo algunas de sus primitivas en los tweets. De este modo, pueden usarse para invocar servicios y coordinar actividades colaborativas. No obstante, los tweets quedan limitados por una sintaxis muy estricta y por lo tanto no se adaptan bien a una comunicación natural humano-máquina. También en esta línea, HyperTwitter [8] propone el uso de Twitter para Ingeniería del Conocimiento colaborativa. Para ello, define una sintaxis de estilo RDF a detectar en los tweets. La Ingeniería del Conocimiento no es el campo que pretendemos abarcar, sino el desarrollo de aplicaciones basadas en mensajes instantáneos; por eso, contamos con que el usuario emplee lenguaje natural, facilitando al desarrollador una sintaxis adecuada para procesar los tweets, las herramientas necesarias para crear aplicaciones basadas en mensajería instantanea, y un entorno avanzado de desarrollo basado en DSDM.

En cuanto al uso de procesamiento de lenguaje natural, [15] propone una técnica híbrida para reconocer entidades con nombre. Otros trabajos, como [2] abarcan el procesado de listados de tweets para verificar si cumplen ciertos criterios. En el futuro, consideraremos la posibilidad de integrar algunas de estas técnicas con nuestro trabajo. Sin embargo, no hay que perder de vista que los tweets están limitados a 140 caracteres ni que los usuarios tienden a utilizar jerga y abreviaciones. Por esto, para poder utilizar técnicas de NLP probablemente sería necesario recurrir a técnicas de normalización como las de [19]. En nuestro caso el trabajo de reconocimiento es mucho más sencillo, ya que definimos los patrones a priori y permitimos variaciones de sus conceptos (como vocales ausentes o el uso de mayúsculas o minúsculas).

Otra línea de trabajo relacionado es la Ingeniería de Aplicaciones Sociales [14, 5, 11, 1]. Una aplicación social usa la infraestructura de las redes sociales para mejorar su alcance. Por ejemplo, en [14] los autores proponen una simbolización para concatenar diferentes acciones a ejecutar sobre los recursos de una red social, como publicar un comentario o subir una foto. Analogamente a Twiagle, su usuario objetivo son los no expertos, y facilita un editor web, evitándoles tareas de programación de bajo nivel. Nuestra solución además permite desarrollar aplicaciones basadas en Twitter y la conexión con fuentes de datos externas.

En suma, asistimos a un interés creciente por el desarrollo de todo tipo de aplicaciones sociales, y el análisis de las interacciones que tienen lugar en las redes sociales. Por eso, creemos que un entorno DSDM como Twiagle ayudaría enormemente al desarrollo de estos dos aspectos, incluso por usuarios con escasos conocimientos en informática.

8 Conclusiones y trabajo futuro

En este artículo hemos introducido el concepto de aplicación basada en mensajes instantáneos, usando Twitter como prototipo. Las entradas y salidas de estas aplicaciones se extraen y producen desde mensajes instantáneos. Hemos estudiado varios escenarios de utilidad, y demostrado la viabilidad de la idea mediante un prototipo (Twiagle), ilustrado con un ejemplo de una votación. La herramienta sigue un enfoque basado en Desarrollo Dirigido por Modelos, y su idea es permitir la creación de aplicaciones basadas en tweets incluso por usuarios inexpertos en informática.

En el futuro, queremos mejorar la expresividad de nuestro lenguaje de patrones, por ejemplo considerando técnicas avanzadas de procesamiento del lenguaje natural. Estamos mejorando el lenguaje de acciones con más primitivas, y el soporte a otras redes sociales. También sería interesante el desarrollo de DSLs sobre Twittern y Twition para aplicaciones específicas, como por ejemplo las basadas en cuestionarios, o de aprendizaje móvil. Finalmente, además de estar mejorando las prestaciones de la herramienta, esperamos realizar una validación empírica con usuarios reales.

Agradecimientos. A los revisores por su detallados comentarios y al Ministerio de Economía y Competitividad (proyecto TIN2011-24139) por la financiación recibida.

References

1. S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar. The jabberwocky programming environment for structured social computing. In *24th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 53–64. ACM, 2011.
2. N. Asadi and J. Lin. Fast candidate generation for real-time tweet search with bloom filter chains. *ACM Transactions on Information Systems*, 31(3):13, 2013.
3. Associated Press. Live action: Twitter grabs super bowl spotlight. http://bigstory.ap.org/article/live-action-twitter-grabs-super-bowl-spotlight, February 2013.
4. B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *21st ACM Symposium on Principles of Database Systems (PODS)*, pages 1–16. ACM, 2002.
5. M. Brambilla and A. Mauri. Model-driven development of social network enabled applications with webml and social primitives. In *ICWE Worksh.*, volume 7703 of *LNCS*, pages 41–55. Springer, 2012.
6. B. P. Carlo Marchetti and P. Plebani. A quality model for multichannel adaptive information systems. In *WWW Alt.'04*, pages 48–54. ACM, 2004.
7. L. Ferrari, A. Rosi, M. Mamei, and F. Zambonelli. Extracting urban patterns from location-based social networks. In *GIS-LBSN*, pages 9–16. ACM, 2011.
8. M. Hepp. Hypertwitter: Collaborative knowledge engineering via twitter messages. In *17th International Conference on Knowledge Engineering and Management by the Masses (EKAW)*, volume 6317 of *LNCS*, pages 451–461. Springer, 2010.
9. P. Hyman. 'Peace technologies' enable eyewitness reporting when disasters strike. *Commun. ACM*, 57(1):27–29, 2014.

10. Ifttt. https://ifttt.com/.
11. N. Kokciyan, S. M. Üsküdarli, and T. B. Dinesh. User generated human computation applications. In *SocialCom/PASSAT*, pages 593–598. IEEE, 2012.
12. J. Krämer and B. Seeger. Semantics and implementation of continuous sliding window queries over data streams. *ACM Transactions on Database Systems*, 34(1), 2009.
13. S. Kumar, F. Morstatter, and H. Liu. *Twitter Data Analytics*. SpringerBriefs in Computer Science. Springer, 2014.
14. J. Laconich, F. Daniel, F. Casati, and M. Marchese. From a simple flow to social applications. In *ICWE Workshops*, volume 8295 of *LNCS*, pages 39–50. Springer, 2013.
15. C. Li, A. Sun, J. Weng, and Q. He. Exploiting hybrid contexts for tweet segmentation. In *SIGIR*, pages 523–532. ACM, 2013.
16. J. Li, A. Vishwanath, and H. R. Rao. Retweeting the Fukushima nuclear radiation disaster. *Communications of the ACM*, 57(1):78–85, Jan. 2014.
17. LibreSoft Gymkhana. http://gymkhana.libresoft.es/, February 2014.
18. G. A. Miller. Wordnet: A lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.
19. J. Porta and J.-L. Sancho. Word normalization in twitter using finite-state transducers. In *Tweet-Norm@SEPLN*, volume 1086 of *CEUR Workshop Proceedings*, pages 49–53. CEUR-WS.org, 2013.
20. E. Qualman. *Socialnomics: How Social Media Transforms the Way We Live and Do Business, 2nd edition*. Wiley, 2012.
21. Reactive manifesto. http://www.reactivemanifesto.org.
22. T. Sakaki, M. Okazaki, and Y. Matsuo. Tweet analysis for real-time event detection and earthquake reporting system development. *IEEE Trans. Knowl. Data Eng.*, 25(4):919–931, 2013.
23. A. M. Segura, J. de Lara, and J. S. Cuadrado. Twiagle: a tool for engineering applications based on instant messaging over twitter. In *International Conference on Web Engineering (ICWE'14)*, volume In Press of *LNCS*, pages 1–4. Springer, 2014.
24. M. Treiber, D. Schall, S. Dustdar, and C. Scherling. Tweetflows: Flexible workflows with twitter. In *Proc. 3rd International Workshop on Principles of Engineering Service-Oriented Systems*, PESOS '11, pages 1–7. ACM, 2011.
25. S. Verma, S. Vieweg, W. . Corvey, L. Palen, J. Martin, M. Palmer, A. Schram, and K. Anderson. Natural language processing to the rescue? extracting "situational awareness" tweets during mass emergency. In *Fifth International Conference on Weblogs and Social Media (ICWSM)*. The AAAI Press, 2011.
26. F. Zhao and A. K. H. Tung. Large scale cohesive subgraphs discovery for social network visual analysis. *Proceedings of the VLDB Endowment (PVLDB)*, 6(2):85–96, 2012.

Simulating Mobile Agents in Vehicular Networks

Óscar Urra, Sergio Ilarri, and Eduardo López

Department of Computer Science and System Engineering
University of Zaragoza
ourra@ita.es, silarri@unizar.es, eduardologa@gmail.com

Abstract. In the next years, vehicular ad hoc networks (VANETs) are expected to become a reality and a great number of interesting applications for drivers and passengers will be developed (related to safety, comfort, and entertainment). In all these applications, the acquisition, management and an efficient and effective exchange of data will be key issues. Therefore, significant data management challenges arise in this context. We argue that mobile agent technology could play an important role as a middleware for the development of applications for vehicular networks, as they naturally support disconnected operations and distributed data management. Overall, a development of solutions based on agents that autonomously take decisions may be promising.
A significant problem arises when we need to evaluate a data management approach for vehicular networks. Deploying it in the real-world in order to evaluate the proposal with real cars is impractical and very expensive, and so field tests are limited to very small-scale and controlled scenarios, mainly as a proof of concept. Instead, simulators are usually used for experimental evaluation. However, existing simulators for vehicular networks do not directly support testing data management techniques based on the use of mobile agents. In this paper, we present a simulator that offers interesting functionalities for that context. The simulator has been developed in a quite generic and extensible way, in order to facilitate its use in a variety of scenarios to test different data management approaches.

Keywords: vehicular networks, mobile agents, simulations, mobile computing, mobile ad-hoc networks

1 Introduction

In the last years, continuous technological advances have lead to the popularity of small computing devices with improved processing and communication capabilities. These devices are portable and have a small size and a moderate price, which has encouraged their adoption and nowadays they are ubiquitous. By exploiting available communication networks (e.g., Wi-Fi hot-spots or mobile phone networks), these devices could access and share a huge amount of data. In particular, *mobile peer-to-peer networks* (*mobile P2P networks*) [1], where the

devices dynamically form an ad hoc network using short-range wireless communications without the need of any support infrastructure, can provide significant benefits over traditional centralized architectures.

When the participating nodes are vehicles traveling along the roads or streets of a certain area, then we have a special case of mobile P2P network, called a *vehicular ad hoc network* (*VANET*) [10]. Vehicular networks are expected to become a reality in the upcoming years and a great number of interesting applications for drivers and passengers will be developed. However, VANETs imply a number of significant challenges from the point of view of data management [3–5], related to the acquisition, processing and the efficient and effective exchange of data among vehicles. Beyond the traditional limitations of mobile computing scenarios (e.g., lower performance in comparison with more powerful desktop computers, changing environment, limited battery lifetime of mobile devices carried by users, etc.), many of the difficulties are a consequence of the use of short-range wireless communication technologies (e.g., about 200 meters), which are unstable, subject to frequent disconnections (network partitioning), have a limited communication range, and are subject to security attacks (since the transmitted data are broadcasted).

From our point of view, the use of mobile agent technology in this environment is a research avenue worth exploring. Mobile agents are autonomous software entities that have the ability to transfer themselves from one execution environment to another by using a communication network. We believe that mobile agents could play an important role as a middleware for the development of applications for vehicular networks, as they naturally support disconnected operations and distributed data management. Overall, they can be an interesting design abstraction in a variety of scenarios.

If we focus on the test phase of the development of applications and data management strategies for vehicular networks, it is clearly very expensive and inconvenient to perform the software testing using real devices in real cars. So, real-world tests are usually limited to very small and controlled scenarios, mainly as a proof of concept or to obtain measures that can be used to fine-tune some simulator. The most logical alternative is indeed to simulate the behavior of the developed software in a controlled and simplified environment that can be managed more easily and that supports a large-scale simulation with a high number of vehicles. There exists a variety of simulation software to test and analyze with the required precision different aspects of communication networks and vehicular traffic. However, as far as we know none of the most popular simulators can be easily used to simulate scenarios with the goal of evaluating data management solutions based on the use of mobile agents. Motivated by this, we have developed a vehicular network simulator in which mobile agents can also be simulated, as well as a set of tools intended to ease the analysis of the results obtained in a variety of scenarios.

The rest of this paper is structured as follows. In Section 2, we briefly describe the technological context of our research area. In Section 3, we focus on the

development of our own simulator, describing its architecture and functionalities. Finally, in Section 4, we summarize our conclusions and some lines of future work.

2 Technological Context

In this section, we present an overview of the most important elements present in the considered scenario: vehicular networks, mobile agents, and simulators.

2.1 VANETs

A *vehicular ad hoc network* (*VANET*) [10] is a highly mobile network whose nodes are vehicles traveling along roads or highways. They can establish connections with other nearby vehicles, and in this way they can exchange relevant information for drivers.

In a vehicular ad hoc network, the vehicles are equipped with short-range wireless communication devices (such as Wi-Fi or UWB) and can establish connections with other nearby vehicles (*vehicle-to-vehicle or –V2V– communications*) in a peer-to-peer way. Using this type of networks in vehicular applications has a number of advantages over a traditional client-server approach, such as: 1) there is no need of a dedicated centralized support infrastructure (expensive to deploy and maintain); 2) the users do not need to pay for the use of these networks; and 3) it allows a very quick and direct (i.e., without intermediate proxies or routers) exchange of information between two vehicles that are within range of each other, which may be critical for safety applications for vehicular networks. Moreover, many application scenarios do not need to communicate with a specific target vehicle but with all the vehicles within a certain area, and therefore broadcast (the typical communication mechanism in VANETs) is a suitable choice.

Other communication schemes can also be considered, based on a fixed infrastructure or mobile telephony networks (e.g., 3G). Thus, even if it may be unrealistic to assume the availability of a generalized wide-area fixed infrastructure in the next years, mobile telephony networks already offer new perspectives for the development of applications to assist drivers. Anyway, such solutions, based on a centralization of the data and decision processes, still suffer from issues such as poor scalability or low reaction time available when dealing with some events like an emergency braking. So, although it is important not to rely on a fixed network infrastructure, which can be difficult and expensive to deploy at a large scale and with global availability, some roads could also offer some static relaying devices which provide Internet access to nearby vehicles by using a fixed network (enabling *vehicle-to-infrastructure –V2I– communications*).

VANETs open up a wide range of opportunities to develop interesting systems for drivers. Although safety applications are usually emphasized, there are also interesting applications related to comfort, entertainment, and travel efficiency. However, a number of difficulties can also arise. Some of these difficulties are due to the fact that two vehicles can communicate directly only if they are

near each other (the range of the wireless communication devices may be limited in practice to about 100-200 meters), which may leave a small time window available for potential data exchange. As the vehicles are constantly moving, the duration during which a communication link is alive can be very short (a few seconds), especially when two nearby vehicles move quickly in opposite directions. Also due to the short communication range, it is possible that there exists no direct connection between two vehicles in the network, in which case the use of some multi-hop communication protocol is necessary. These protocols are usually complex and it is difficult to guarantee an upper bound on the amount of time needed to deliver a message to a recipient, due to the fact that the existing links change constantly.

The development of applications for vehicular networks requires taking these constraints into account. Besides, they may need to consider also other special features of this dynamic context: the geographic area of interest could spread over a large extension, users (usually, the drivers) can move constantly at different speeds and directions (sometimes according to predefined routes, but not necessarily), the density of the network nodes can vary depending on the place or the time of the day, etc.

2.2 Mobile Agents

Mobile agents are software entities that run on an execution environment (traditionally called *place*), and can autonomously travel from *place* to *place* (within the same computer or between different computers) and resume their execution at the target [9]. Thus, they are not bound to the computer/device where they are initially created and they can move freely among computers/devices. To be able to use mobile agents it is necessary to execute a middleware known as a *mobile agent platform* [11], which provides agents an environment where they can execute as well as the ability to move to other execution environments and other services (e.g., communication, security, persistence, etc.). This platform must be executed on every computer or device that could host agents in the distributed system, and it must offer some interface to communicate with other platforms present in other devices or computers. Key functions provided by a mobile agent platform is its ability to locate other available platforms and transmit the code of an agent from the origin to the destination of the agent's movement.

Thanks to the mobility capability of mobile agents, it is easy to build complex distributed applications that are at the same time flexible. Thus, a mobile agent can carry a required task wherever it is needed. If the task executed by an agent must be changed in the future, a new version of the agent (a new agent implementation) can be delivered. Thus, there is no need to keep specialized software installed on the computers/devices composing the distributed system: only the generic mobile agent platform software is needed and an agent implementing the required behavior can move there at any time.

Mobile agents can be designed and programmed to provide interesting benefits (e.g., autonomy, flexibility, and effective usage of the network) that make them very attractive for distributed computing. Particularly, and motivated by

the increasing popularity of mobile devices, mobile agents have been found useful for the development of applications in mobile environments. A mobile environment has a number of special properties, such as the need to rely on wireless communications due to the mobility of the mobile devices, that create a scenario completely different from that of a traditional distributed environment with fixed networks. Such an environment has a number of advantages (e.g., the processing is not tied to a fixed location) but also some drawbacks, such as the limited computational power of mobile devices and the communication constraints imposed by the use of wireless communications (that usually either offer a low bandwidth, a high latency, and intermittent/unreliable connectivity, or are expensive or not available everywhere).

The autonomy, intelligence, and movement capabilities of mobile agents render them a powerful and flexible tool to build distributed systems, especially in mobile environments. For example, a mobile agent could be programmed to visit certain devices in a complex network whose nodes are mobile devices and (once it reaches devices storing relevant data) to process the local data available there in order to collect new interesting data, and finally to return to the origin with the final relevant data collected. So, mobile agents can move the processing to the data source instead of bringing all the data to the node that will perform the processing (thus reducing the amount of data communicated). Other interesting advantages of mobile agents include their ability to support disconnected operations (an agent can live outside its *home device*, which can be turned off while the agent is performing its tasks elsewhere), to exploit the most suitable resources available (e.g., using powerful fixed computers instead of the limited resources of a mobile device, when appropriate), and to minimize network connections (as opposed to a traditional client/server approach, that requires a connection open and alive while the request is being performed, an equivalent request processing using mobile agents would only require the connection active during the movements of the agents).

2.3 Using Mobile Agents in VANETs

As we have seen, VANETs offer interesting opportunities for the development of applications for vehicles. However, they also introduce some difficulties. For example, in VANET applications, data may need to be transported from vehicle to vehicle in order to reach locations that are not directly accessible due to the short range of the wireless communications used. Then, two major problems arise. First, as the propagation of the data can be slow, the information can be outdated when it reaches its destination. Second, it can be difficult to determine the destination itself and how to reach it. Given the highly mobile nature of a VANET, the destination could be a single specific vehicle, every vehicle present in a geographic area, all those vehicles matching a certain condition, etc.

To deal with these drawbacks, mobile agents can be very useful because of their adaptability and mobility features. Thus, *they can bring a processing task wherever it is needed*, and *the algorithm or agent's logic can be changed at any time* by deploying new versions of the agent code. This flexibility is

quite interesting in a vehicular network. A mobile agent-based application for a vehicular network can be updated by just releasing new versions of the involved agents, without the need to upgrade the software system of all the vehicles.

Another important advantage of mobile agents in vehicular networks is that *they can move to wherever the data are located in order to process and collect only the relevant data* (filtering out data which may be unnecessary). For example, if we want to obtain some information from vehicles located within a certain geographic area, a mobile agent could move there and process the data locally. Once the most interesting data are obtained, they will be carried along with the mobile agent, keeping their size smaller (irrelevant data are discarded), and making it easier to transmit them in a scenario where communications could be constrained.

Finally, we argue that mobile agents can be *very useful for data dissemination* in vehicular networks. Thus, they can adapt easily to changing environmental conditions in order to improve the dissemination. For example, a basic flooding dissemination protocol will fail if the traffic density of the vehicles is low and there are not enough vehicles to re-diffuse the data towards the target, as well as other problems such as the *storm broadcast* (network overloading). Other dissemination protocols, such as carry-and-forward [15], where the vehicles may hold the data to be transmitted until these data can be relayed to other vehicles, can be used in the case of low traffic density. However, considering the variety of existing dissemination protocols (and others that could be developed in the future), mobile agents seem an ideal technology to implement flexible and dynamic dissemination approaches and take suitable routing decisions. In this way, an agent can carry data and decide where and when to move, whether it should wait in the current vehicle before jumping to another one, whether it could be beneficial to clone itself, etc. With this approach, the routing decisions lie with the data themselves (encapsulated in the mobile agents) and different dissemination protocols (dynamic and adaptive to the current conditions) can be implemented.

2.4 Simulators for VANETs

A significant number of network and traffic simulators have been developed, both commercial and free or open source. In this section we briefly present some of the most popular ones.

Network Simulators. Network simulators allow the configuration and simulation of detailed parameters of the devices and the communication process. For example, the technology used for data transmission (wireless, copper wire, fiber optic, etc.), the data loss ratio, latencies, shadowing effects that make wireless communications more difficult, distance attenuation, etc. Some of the most used simulators of this type are *NS-3* and *Qualnet* (*Quality Network*). In our simulation approach we simulate communication networks from a high-level perspective, so we will not describe these simulators in more detail.

Traffic Simulators. Traffic simulators are specialized in the movement of the vehicles and allow to generate traces of their movement, following different patterns and behaviors in different scenarios. Some examples of such simulators are SUMO and VanetMobiSim.

SUMO (*Simulation of Urban MObility*)[1] is an open source microscopic road traffic simulator. It allows the simulation of vehicles as single entities, with the ability of traveling through specific routes, changing the road lane, and following the traffic rules. It can handle scenarios with large road networks and a high number of vehicles. It can be enhanced with plugins and can interoperate with other software both by importing and exporting data using different file formats.

VanetMobiSim[2] focuses on vehicular mobility, and features realistic automotive motion models at both macroscopic and microscopic levels. At macroscopic level, it can import maps from the US Census Bureau database, or randomly generate them using a Voronoi tessellation. It has also support for multi-lane roads, separate directional flows, differentiated speed constraints, and traffic signs at intersections. At microscopic level, it implements different mobility models, providing realistic car-to-car and car-to-infrastructure interaction. According to these models, vehicles regulate their speed depending on nearby cars, overtake each other, and act according to traffic signs in the presence of intersections.

Hybrid Simulators. A hybrid traffic-network simulator can simulate both traffic and network elements in a geographic scenario. Some examples of such simulators are EstiNet and VEINS.

EstiNet[3] is a commercial product consisting on an extensible network simulator and emulator capable of simulating various protocols used in both wired and wireless IP networks, as well as wireless vehicular networks (including V2V and V2I communications), among others. Regarding its traffic simulation capabilities, it can simulate multi-lane road networks, it incorporates different microscopic vehicle mobility models, and the behavior of any vehicle can be changed as it receives messages from the vehicular network. It has been used for modeling VANETs and other ad-hoc networks as well as for the evaluation of real-life P2P applications and traffic signal control algorithms.

VEINS (*Vehicles in Network Simulator*)[4] is an open source software that supports online re-configuration and re-routing of vehicles in reaction to network packets, it supports different vehicular mobility models, and relies on detailed models of IEEE 802.11p and IEEE 1609.4 DSRC/WAVE network layers (including multi-channel operation, QoS channel access, noise and interference effects). It can import scenarios from OpenStreetMap[5], including buildings, speed limits, lane counts, traffic lights, and access and turn restrictions. It can also employ validated and computationally inexpensive models of shadowing effects caused

[1] `http://sumo-sim.org/`
[2] `http://vanet.eurecom.fr/`
[3] `http://www.estinet.com`
[4] `http://veins.car2x.org`
[5] `http://www.openstreetmap.org`

by buildings as well as by vehicles. Finally, it supplies data sources for a wide range of metrics, including travel time and vehicle emissions.

It is also interesting to mention that the use of videogames to facilitate the evaluation of data management approaches for vehicular networks has also been proposed recently (see the *VANET-X* videogame[6]).

3 A Simulator for VANETs with Mobile Agents

Network simulators have very limited (or non-existing) capabilities to simulate moving objects such as vehicles, whereas traffic simulators cannot simulate network communications, although some of them can export mobility traces to be imported later in a network simulator. A hybrid simulator can simulate both aspects at the same time.

However, none of the simulators mentioned can directly support the simulation of mobile agents, and so they cannot be easily used to evaluate data management approaches based on mobile agent technology. As this is the focus of our research, we were compelled to develop our own simulator (*MAVSIM*, *Mobile Agents in VANETs SIMulator*), that offers interesting functionalities for that context. The simulator has been developed in a quite generic way, with several configurable parameters and a modular and extensible architecture, to facilitate its use in a variety of scenarios to test different data management approaches.

In the rest of this section, we describe the main features of the simulator developed, its architecture, and a use case. The web site of the simulator (`http://sid.cps.unizar.es/MAVSIM/`) offers additional information, screenshots, videos, and the possibility to download it.

3.1 Features

The main features of MAVSIM are the following:

- It is written in Java, which makes it portable among different architectures and operating systems. It has been successfully tested in Microsoft Windows 7 and 8 (32 and 64 bits), Linux (32 and 64 bits), and Solaris (64 bits).
- It can run both in graphic interactive mode with a Graphical User Interface (GUI), or in batch mode (useful to execute a large number of simulations or experiments in an easy way).
- Simulations can be recorded an replayed later (with step by step, pause, and rewind and forward functionalities), which facilitates a careful analysis of the whole process. For this purpose, the replay tool shown in Figure 1 is used; in that screenshot, we also show in the screen some information that is relevant for the monitoring task scenario described earlier.
- Any road map can be downloaded from OpenStreetMap. There is no limitation in the type of layout imported (cities, highways, rural areas, etc.).

[6] `http://sid.cps.unizar.es/Vanet-X/`

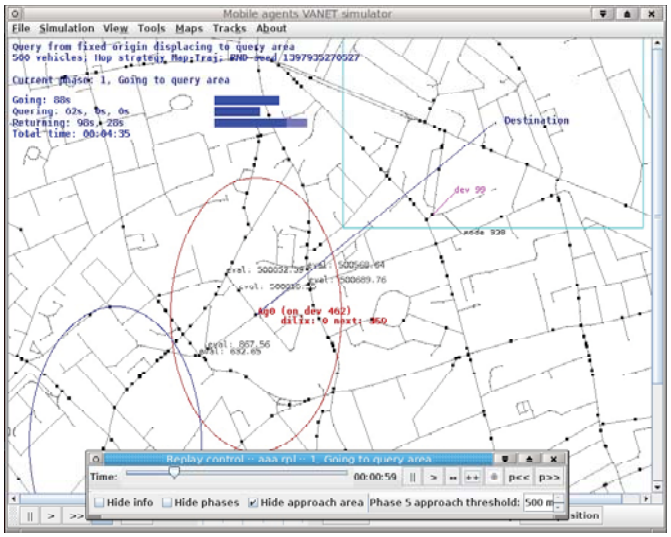


Fig. 1. The replay tool

- The simulated mobile agents can be programmed in a similar way as they would be programmed using real platforms such as SPRINGS [7] (e.g., we provide methods such as *moveTo(targetDevice)* to simulate the movement actions performed by the agents). A built-in generic mobile agent platform is included in the simulator with the most common methods and primitives necessary for this.
- It can import traces generated by real vehicles or by other traffic simulators, and use them for the experiments with mobile agents.
- Road side units (or fixed communication devices) can be simulated.
- It includes a variety of movement algorithms for the simulated vehicles, such as *Random way-point*, *Gauss-Markov*, and others.
- In urban areas it can simulate the presence of buildings that block the wireless signal constraining the communications. For this purpose, the geometric method described in [8] is used.
- The initial scenario conditions for the experiments can be set randomly or using a known *seed*, which allows to reproduce the same conditions and repeat exactly the same experiment (with the same trajectories for the vehicles and other random conditions) if necessary.

A number of parameters can be set to configure in detail the simulation scenario. Table 1 shows a summary of some of the general parameters that can be configured, as well as parameters that are applicable for the specific case of the evaluation of monitoring approaches based on mobile agents. Most parameters, when omitted, take a default value; others are optional and do not have a default value.

Table 1. Summary of configuration parameters

Parameter	Description	Default
General configuration parameters		
-mcr	Mobile communication range	250 m
-v	Number of vehicles to simulate	100
-vld	Vehicle Linear Density (vehicles/km). Overrides -v	
-s	Average speed of vehicles (km/h)	50
-lat	Latency for a mobile agent's trip	1 s
-errRate	Error rate for communications	0%
-batch	Execute in batch mode	false
-map	Load a scenario map	Last used
-mbx	Number of *mailboxes* connected through a wired network (static nodes with storage capacity and the capability to provide wide-area network coverage)	0
-f	Number of fixed (non-moving) wireless devices	1
-mob	Mobility strategy for vehicles (1 = Random, 2 = Routes, 3 = Straight, 4 = Heuristic & Destination, 5 = GaussMarkov, 6 = ManhattanDynamic)	2
-nobldg	Do not simulate buildings in the map as communication obstacles	false
-rec	Record the experiment to a file for later replay and analysis	
-seed	Seed for random numbers (0 for random seed)	0
Configuration parameters to evaluate monitoring approaches		
-d	Initial distance (in meters) to the interest area	1000
-j	Mobile agent hop strategy (1 = RND, 2 = BEP, 3 = Approach, 4 = ANG, 5 = MAP, 6 = EP, 7 = EUC, 8 = Map-Traj, 9 = Optimal)	8
-m	Maximum number of simulation iterations during which to collect data	899
-mih	Minimum expected improvement to enable an agent's hop	0%
-dimArea	Interest area dimensions (height, width, longitude, and latitude: -mzh, -mzw, -mzw, -mzy)	
-prd	Probability that a device contains relevant data / appropriate sensors	50%
-rph	Max distance to hop (as a percentage of the communication range)	100%
-snru	Selection of nearest *Roadside Unit* (0 = none, 1 = once, 2 = continuous)	0

3.2 Structure and Classes

The simulator is designed to be extensible. It contains a number of classes with different functionalities (see Figure 2), and the classes are organized in modules (*packages*): one package contains everything related to graphics, other the functionalities related to mobility strategies, etc. In this way it is easier to maintain the code and perform changes. Moreover, to facilitate its extensibility, some abstract classes and interfaces have been defined, that allow to add new functionalities in a quick and easy way. For example, if it is necessary to add a new mobility strategy for vehicles, there is an abstract class *Mobility* available that includes the necessary methods and attributes for all the mobility strategies and makes adding a new one easier: it would be enough to define a new subclass implementing the methods of *Mobility* and extend the configuration files to add support to select and use that new mobility strategy in the simulations. In the same way, there also exists an abstract class (*TrackReader*) to read different file formats of mobility traces, so if a new format is needed only the appropriate subclass must be implemented. Finally, one particularly important abstract class and interface are *Agent* and *IAgent*, respectively; they are intended to allow the programming of new different types of simulated mobile agents.

In this way, the simulator developed can be easily extended in a number of ways, such as: to incorporate new behaviors for the vehicles (e.g., new mobility strategies), to define new types of mobile agents with different behavior (e.g., different agent mobility strategies), to define different phases in the data management approach for easy analysis of each of them with the replay tool (e.g., in the monitoring task scenario phases such as "going to the area", "measuring environment data", and "coming back to the query originator" are defined, as shown in the upper-left part of Figure 1), or to add new simulation parameters.

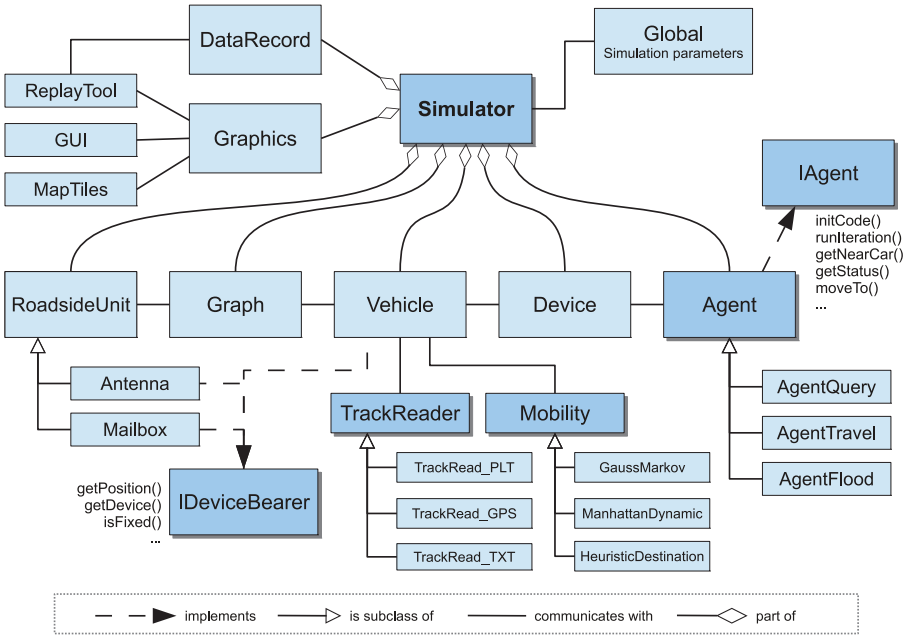


Fig. 2. High-level overview of the architecture of the simulator

We could also simulate data management approaches that do not use mobile agents, as we can simulate that there is a single static agent (i.e., a mobile agent with no mobility) in each equipped vehicle, that implements the required data management functionalities that would be available in the equipped vehicles. Most extensions require defining appropriate subclasses and configuration parameters. Of course, for a comfortable use of the extensions in an interactive simulation mode, changes to the graphical user interface may also be required.

3.3 Example of Simulation Scenario

As an example, we have used this simulator to evaluate *environment monitoring strategies using mobile agents*. The idea is to exploit sensors available in normal vehicles to monitor the environment, using mobile agents to find the relevant cars [13]. In these scenarios, there is a *target area* (or *interest area*) that has to be monitored (e.g., see Figure 3, which shows the target area as a rectangle near the center of the map, the node requesting the monitoring task as a point, and its communication range as a circle around the query point). One or more mobile agents can be sent to that area so that they can use sensors available in the vehicles therein to take measures of some environmental parameter (e.g., the CO_2) in the area. So, conventional vehicles with the appropriate sensors are exploited in a dynamic way by using mobile agents, leading to a more flexible and economic solution than an alternative and more traditional approach where

static sensors are manually deployed within the area of interest. However, the approach based on mobile agents faces some challenges, as the agents need to reach the target area by hopping from car to car (*transportation via communication*, using short-range wireless devices) and/or benefiting from the physical mobility of the vehicles (*transportation via locomotion*), keep themselves in vehicles within the area (to take the required measures), and finally return the results to the query originator. So, a key issue is to develop appropriate *hopping strategies* for the agents.

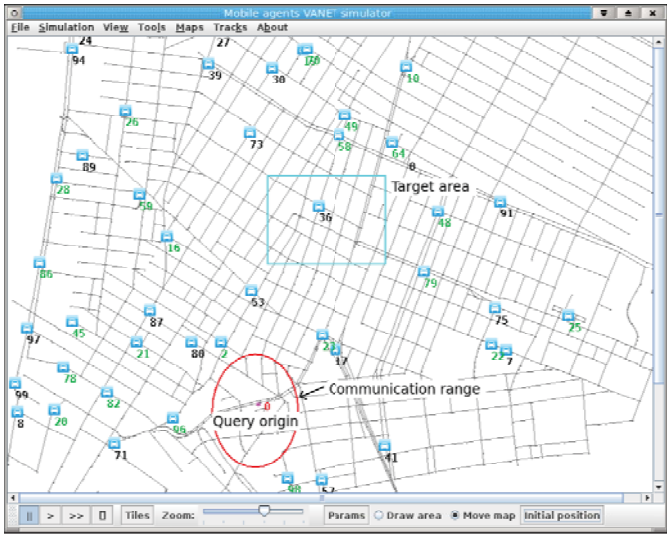


Fig. 3. Simulating agents in a monitoring task: target area and query originator

We want to evaluate the performance of these strategies to know which one is the best, under what circumstances, and how long takes the entire monitoring process to complete. The use of MAVSIM eases the process of designing and testing the behavior of the different strategies and the agents' themselves, thanks to the tight integration of all the simulated objects, their interactions (the *communicates with* in the class diagram of Figure 2), and the facilities available for recording their status at every iteration of the process (e.g. the device and vehicle where the agent is, if an agent hop ends successfully or fails, etc.). In this way, they can be analyzed later and/or reviewed with the replay tool, searching for any anomaly or pattern. If it were necessary to correct the behavior of the simulated agents, given the modularity of the simulator, the only part necessary to be modified would be the agent code. The rest of the elements (vehicles, devices, etc.) would remain the same.

Moreover, when a simulation ends different data are provided: the total number of iterations completed; if the simulated process ended successfully or not;

the number of iterations taken at every phase of the process; the number of times that every mobile agent hopped; their effective speed, etc. All these data can be stored in a database or a spreadsheet, and if the simulation is repeated many times varying the initial conditions, all the data collected can be analyzed to extract conclusions about the whole process performance. Since all the simulation parameters can be set from the command line and be executed unattended in batch mode, it is easy to automatize the execution of a high number of simulations varying only a few initial parameters, making the experimental data collection task very convenient.

4 Conclusions and Future Work

Vehicular networks bring important challenges for the data management community. For practical reasons, the proposed communication and data management strategies are usually evaluated using simulators, which provides flexibility in defining the required test conditions and enables quick results considering large-scale simulated scenarios with an affordable cost. However, although the use of mobile agents in the context of VANETs could be interesting, existing simulators cannot be easily adapted to evaluate approaches based on the use of mobile agents. For that reason, we have developed our own simulator.

We think that the simulator developed is quite well suited for the simulation of mobile agents in vehicular networks, taking into account different elements such as the network communications, the moving vehicles, the presence of buildings that block the signal, and other parameters of interest for testing different scenarios. The simulator is written in a modular and extensible way, so it can be enhanced with new features with little effort. We are currently using the simulator in our research, and we are constantly adding new features when we need them to evaluate different data management strategies. Thanks to the use of this simulator, we have obtained promising results regarding the potential interest of using mobile agents in vehicular networks [12–14].

There are however a number of features and enhancements that have not been implemented so far. Among these, we are especially interested in the development of some extensions to simulate *events* (e.g., accidents, traffic jams, etc.) and *parking spaces*, as they will enable testing data management strategies for the exchange of interesting events for drivers [2] and applications to help drivers in the search of appropriate parking spaces [6]. Moreover, we also plan to study the possibility of mixing the simulator developed with functionalities provided by existing general-purpose traffic simulators (in particular, *SUMO* with *TraCI*).

Acknowledgments This work has been supported by the CICYT project TIN2010-21387-C02-02 and DGA-FSE.

References

1. A. Bonifati, P. K. Chrysanthis, A. M. Ouksel, and K.-U. Sattler. Distributed databases and peer-to-peer databases: past and present. *SIGMOD Record*, 37:5–11, 2008.
2. N. Cenerario, T. Delot, and S. Ilarri. A content-based dissemination protocol for VANETs: Exploiting the encounter probability. *IEEE Transactions on Intelligent Transportation Systems*, 12(3):771–782, 2011.
3. T. Delot and S. Ilarri. Data gathering in vehicular networks: The VESPA experience (invited paper). In *Fifth IEEE Workshop On User MObility and VEhicular Networks (ON-MOVE 2011), at LCN 2011*, pages 801–808. IEEE, October 2011.
4. T. Delot and S. Ilarri. Introduction to the special issue on data management in vehicular networks. *Transportation Research Part C: Emerging Technologies*, 23:1–2, 2012.
5. T. Delot and S. Ilarri. The VESPA Project: Driving advances in data management for vehicular networks. *ERCIM News*, (94):17–18, 2013. Special Theme on Intelligent Vehicles as an Integral Part of Intelligent Transport Systems.
6. T. Delot, S. Ilarri, S. Lecomte, and N. Cenerario. Sharing with caution: Managing parking spaces in vehicular networks. *Mobile Information Systems*, 9(1):69–98, 2013.
7. S. Ilarri, R. Trillo, and E. Mena. SPRINGS: A scalable platform for highly mobile agents in distributed computing environments. In *Fourth International Workshop on Mobile Distributed Computing (MDC'06), at WoWMoM'06*, pages 633–637. IEEE, 2006.
8. F. J. Martinez, M. Fogue, C. K. Toh, J.-C. Cano, C. T. Calafate, and P. Manzoni. Computer simulations of VANETs using realistic city topologies. *Wireless Personal Communications*, 69(2):639–663, 2012.
9. D. Milojicic, F. Douglis, and R. Wheeler. *Mobility: processes, computers, and agents*. ACM, 1999.
10. S. Olariu and M. C. Weigle, editors. *Vehicular Networks: From Theory to Practice*. Chapman & Hall/CRC, 2009.
11. R. Trillo, S. Ilarri, and E. Mena. Comparison and performance evaluation of mobile agent platforms. In *Third International Conference on Autonomic and Autonomous Systems (ICAS'07)*. IEEE, 2007.
12. O. Urra and S. Ilarri. Using mobile agents in vehicular networks for data processing. In *14th International Conference on Mobile Data Management (MDM 2013)*, volume 2, pages 11–14. IEEE, 2013.
13. O. Urra, S. Ilarri, T. Delot, and E. Mena. Using hitchhiker mobile agents for environment monitoring. In *Seventh International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS'09)*, pages 557–566. Springer, 2009.
14. O. Urra, S. Ilarri, T. Delot, and E. Mena. Mobile agents in vehicular networks: Taking a first ride. In *Eight International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2010)*, volume 70 of *Advances in Intelligent and Soft Computing*, pages 118–124. Springer, April 2010.
15. J. Zhao and G. Cao. VADD: Vehicle-assisted data delivery in vehicular ad hoc networks. In *25th IEEE International Conference on Computer Communications (INFOCOM'06)*. IEEE, 2006. 12 pages.

Context-Aware Recommendations in Mobile Environments

María del Carmen Rodríguez-Hernández and Sergio Ilarri

Department of Computer Science and Systems Engineering
University of Zaragoza, Zaragoza, Spain
{692383,silarri}@unizar.es

Abstract. Traditional recommendation systems offer relevant items (e.g., books, movies, music, etc.) to users, but they are not designed for mobile environments. In those environments, the context (e.g., the location, the time, the weather, the presence of other people, etc.) and the movements of the users may be important factors to obtain relevant and helpful recommendations. The emergence of context-aware recommendation systems has prompted the growth of recommendation algorithms that incorporate context information.
However, most existing research in this field considers only static context information, despite the fact that exploiting dynamic context information would be very helpful in mobile computing scenarios. Moreover, the design and implementation of generic frameworks to support an easy development of context-aware recommendation systems has been relatively unexplored. In this paper, we present our ongoing work to develop a context-aware recommendation framework for distributed and mobile environments, which will allow suggesting relevant items to mobile users.

Keywords: context-awareness, recommendation systems, mobile computing.

1 Introduction

Recommendation Systems (*RS*) are applications which suggest (relevant) items to users. The suggestions can help to solve certain decision-making problems which are presented to the users, such as which books to buy, which movies to watch, or which online news to read. They try to adapt the suggestions to each user individually, based on his/her preferences. Depending on how the recommendations are obtained, a recommendation system can be classified in three main categories [1]: *content-based*, based on *collaborative filtering*, and *hybrid*.

Traditional RS operate in a *two-dimensional* (2D) $User \times Item$ space. Moreover, generally, traditional collaborative filtering techniques take into account all the collected ratings of the items to generate the recommendation models. These techniques assume that the context is homogeneous, but actually a user can assign different ratings to the same item in diverse contexts, as the relevance and interest of a specific data item for a user may depend on his/her current context.

Motivated by that observation, more modern RS consider contextual information in the recommendation process. Examples of contextual information are the location, the time, the weather, the user's current activity, and the network/communication capabilities. A pioneer proposal in the field of context-aware recommendation systems is [2], which advocates using context information to improve the recommendations. To do so, the authors extended the classical 2D paradigm to a *multidimensional recommendation model* (*MD model*) that provides recommendations based on multiple dimensions: $User \times Item \times Context$. RS that incorporate context information in the recommendation process were denominated *Context-Aware Recommender Systems* (*CARS*).

In this paper, we present our undergoing work regarding the development of a new context-aware recommendation framework for mobile environments, which will support the suggestion of useful recommendations to mobile users. The rest of the paper is organized as follows. In Section 2, we introduce the scientific problem domain addressed. Section 3 summarizes some studies related to the topic of context-aware recommendations. In Section 4, we describe the methodological approach proposed to solve and validate the solution to the problem presented. In Section 5, we explain the research that we have carried out so far. Finally, the expected contributions of this work are provided in Section 6.

2 Problem Domain

Context-Aware Recommender Systems (*CARS*) is still an emerging field. Indeed, Adomavicius and Jannach claimed recently that there is still much research needed in the field of CARS [3]. Most researchers so far have worked on research issues related to the problem of understanding and representing the context in the recommender systems (e.g., [2]). So, they basically have tried to determine how the context could be modeled in recommendation systems. Other researchers have proposed different recommendation algorithms which include context information in the recommendation process (e.g., [4, 5]). Recently, in [6] the authors introduced a software framework to build complex context-aware applications. However, there is still significant work that needs to be done in this area. So, despite these efforts, the design of flexible and generic architectures and frameworks to support an easy development of CARS has been relatively unexplored, particularly if we consider mobile computing environments.

Based on the above issues, we focus on the following scientific problem: *how could we facilitate the development of CARS in distributed environments to provide mobile users with relevant recommendations?* So, the object of our research study is the process of context-aware recommendation for mobile users. Specifically, we pursue the following general goal: the development of a mobile context-aware framework which will allow providing relevant recommendations to mobile users. We aim at generic mobile computing scenarios where different communication technologies (e.g., 3G, WiFi), interaction protocols (centralized approaches vs. mobile peer-to-peer), and data access modes (push-based data access, pull-based data access, and hybrid data access) may be available. Moreover, the

relevant data used as a basis for the recommendations may be available locally on the mobile device, centralized on a server, or distributed among different data sources (mobile devices and other static nodes). The recommendation framework should thus adapt itself to different typical mobile computing scenarios.

3 Related Work

The interest of incorporating contextual information into recommendation systems has been emphasized in studies such as [2]. In [7], the authors proposed three recommendation paradigms: *pre-filtering*, where the contextual information is used to filter the data set before applying traditional recommendation algorithms; *post-filtering*, where the ratings are predicted using a conventional 2D recommendation system, taking all the input data available into account, and then the resulting set of recommendations is adjusted (contextualized) for each user by using contextual information; and *contextual modeling*, which uses the context information directly in the modeling technique.

The models proposed in [8, 9] are examples of the contextual modeling paradigm, but these proposals model information in very specific application domains, and so their cannot be easily reused in other recommendation scenarios. There are also some proposals that try to exploit available contextual information and be more generic, but they have some limitations. Thus, in [10] the authors emphasize that those approaches usually represent information that either concerns particular application domains (e.g., tourism, movies, etc.) or more abstract domains (e.g., products, web services, e-learning, etc.), but that a truly generic contextual model for CARS was missing, which motivated their work. As an example, [11] presents a generic model using an ontology, which can be used in different types of recommender systems and models data, context, and the recommendation process itself. Recently, [10] carried out a study to try to determine whether a more generic modeling approach could be applied for CARS. As a result of the study, the authors proposed a novel generic contextual modeling framework for CARS, which was theoretically evaluated with positive results.

Particularly in mobile environments, where the user is moving and the context is highly dynamic, it is important to provide precise recommendations and avoid overloading the user with the suggestion of many items. Generally, mobile devices such as smartphones have important limitations in comparison to traditional mobile or desktop computers. For example, they usually provide restricted input facilities (e.g., lack of a comfortable keyboard). Therefore, a recommendation system should try to relieve the user from having to type or introduce significant information as an input, favoring implicit recommendations (based on the context and user preferences) over explicit (query-based or user-initiated) recommendations. Along these lines, [12] proposes a proactive recommender system that pushes recommendations to the user when the current situation (i.e., the context) is considered appropriate, without explicit user requests.

A new class of ubiquitous context-aware recommendation systems, called *UbiCARS*, has been proposed in [13]. The idea is to combine characteristics of

both ubiquitous systems and CARS. Systems in this category are ubiquitous in the sense that they capture information from the environment and react on it. At the same time, they are context-aware because they consider the context in the recommendation process by using multidimensional contextual datasets.

Some open research problems for CARS, identified in [14], have been addressed to a certain extent in recent studies. For example, in [15] the authors performed an evaluation and comparison of the effectiveness (accuracy vs. diversity) of existing paradigms, in order to identify strengths and weaknesses of each paradigm and to determine which one is better in different circumstances. Moreover, the inclusion of *diversity* as an important element in context-aware recommendation systems was considered for the first time in [16]: the idea is that the users should be provided with recommendations that are diverse enough rather than very similar to each other, which is an idea that had been exploited before in Information Retrieval (IR) contexts [17].

Recently, a software framework called *Hybreed* [6] has been presented for building complex context-aware applications. The framework is based on a quite generic notion of context. Its creators actually solved some of the open research problems identified in [14]. However, the authors identified elements pending for future work, such as developing and providing advanced learning algorithms as part of the framework, the development of a processing engine that supports distributed and asynchronous workflows, and addressing privacy issues.

Based on the above, there is a need to define a generic and abstract framework for the modeling of CARS in mobile environments. With our work, we attempt to bring recommendation systems to dynamic mobile computing environments.

4 Methodological Approach

To accomplish our goal, the first step is to perform a detailed study of the state of the art on context-aware recommendation systems, as well as an analysis of existing frameworks which may solve some of the problems arising in this field. The next step is to develop an appropriate and general architecture that supports context-aware recommendations in generic mobile environments. Then, a framework implementing that architecture will be developed using Java and exploiting useful class libraries that facilitate the use of recommendation models (e.g., Mahout) and machine learning techniques (e.g., Weka).

On the other hand, we will evaluate and validate the proposed framework by considering several application scenarios. Unfortunately, most existing datasets used to evaluate traditional recommendation algorithms do not consider the contextual information or the existence of mobile users. So, an important difficulty that we will have to face during the experimental evaluation of the framework will be the collection of appropriate data sets for CARS in mobile environments.

Moreover, performing experiments with real mobile users and mobile devices (e.g., mid-range smartphones) would be expensive and impractical. Therefore, we will require the use or the development of a simulator that should be able to model both mobile users and items, as well as their context. Moreover, the

suggestion of items to mobile users could affect their behavior and mobility (e.g., in case a user decides to move towards an item recommended).

5 Research Carried Out

So far, we have performed an initial study of the state of the art on context-aware recommendation systems. We identified that most research focuses on the problem of understanding and representing the context. So, existing proposals have mainly tried to determine how the context could be modeled in recommendation systems. There are also some proposals of recommendation algorithms which include contextual information in the recommendation process. However, the design of generic architectures and frameworks to support an easy development of CARS has received little attention, especially in mobile environments.

We have also started the development of a generic context-aware mobile recommendation architecture. It will include several traditional recommendation algorithms (e.g., *collaborative filtering based on users/items* and *content-based*), as well as hybrid algorithms that combine several of them to try to improve the accuracy of the recommendations. Furthermore, it will provide different context-aware recommendation paradigms (e.g., *pre-filtering*, *post-filtering*, and *contextual modeling*). The recommendation process will consider both static context information obtained from the user's profile as well as dynamic context data obtained from the environment (e.g., by using sensors). Moreover, it will support the exploitation of other external data sources available, such as geospatial information services, social networks, and web services.

The user will be able to receive both proactive recommendations (real-time recommendations of items without explicit requests from the user) and reactive recommendations (obtained as an answer to a query submitted by the user and evaluated by the system as a continuous query). Different types of environments, including *outdoor* environments (e.g., roads, pedestrian paths) and *indoor* environments (e.g., a mall, a hotel, a supermarket), will be supported.

6 Expected contributions

The ultimate goal of this research is to develop a framework that will provide relevant context-aware recommendations for mobile users. We will also probably need to develop a suitable simulator that will allow to represent mobile users that receive real-time context-aware recommendations in both indoor and outdoor scenarios. An intermediate contribution will be the design of a generic architecture to support an easy development of context-aware recommendation systems for mobile users. Overall, with this research, we hope to contribute to bridge the gap between recommendation systems and mobile computing.

Acknowledgments

This work has been supported by the CICYT project TIN2010-21387-C02-02, DGA-FSE, and a Banco Santander scholarship held by the first author.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering **17**(6) (2005) 734–749
2. Adomavicius, G., Sankaranarayanan, R., Sen, S., Tuzhilin, A.: Incorporating contextual information in recommender systems using a multidimensional approach. ACM Transactions on Information Systems **23**(1) (2005) 103–145
3. Adomavicius, G., Jannach, D.: Preface to the special issue on context-aware recommender systems. User Modeling and User-Adapted Interaction **24**(1-2) (2014) 1–5
4. Oku, K., Nakajima, S., Miyazaki, J., Uemura, S.: Context-aware SVM for context-dependent information recommendation. In: Seventh International Conference on Mobile Data Management, IEEE Computer Society Washington (2006) 1–4
5. Ricci, F., Baltrunas, L.: Context-based splitting of item ratings in collaborative filtering. In: Third ACM conference on Recommender systems, ACM (2009) 245–248
6. Hussein, T., Linder, T., Gaulke, W., Ziegler, J.: Hybreed: A software framework for developing context-aware hybrid recommender systems. User Modeling and User-Adapted Interaction **24**(1-2) (2014) 121–174
7. Adomavicius, G., Tuzhilin, A.: Context-Aware Recommender Systems. In: Recommender Systems Handbook. Springer (2011) 217–253
8. Santos, O., Boticario, J.: Modeling recommendations for the educational domain. Procedia Computer Science **1**(2) (2010) 2793–2800
9. Sielis, G., Mettouris, C., Papadopoulos, G., Tzanavari, A., Dols, R., Siebers, Q.: A context aware recommender system for creativity support tools. Journal of Universal Computer Science **17**(12) (2011) 1743–1763
10. Mettouris, C., Papadopoulos, G.: Contextual modelling in context-aware recommender systems: A generic approach. In: Web Information Systems Engineering (WISE 2011 and 2012) Workshops. (2013) 41–52
11. Loizou, A., Dasmahapatra, S.: Recommender systems for the semantic web. In: ECAI 2006 Recommender Systems Workshop. (2006)
12. Woerndl, W., Huebner, J., Bader, R., Gallego-Vico, D.: A model for proactivity in mobile, context-aware recommender systems. In: Fifth ACM Conference on Recommender Systems (RecSys 2011), ACM (2011) 273–276
13. Mettouris, C., Papadopoulos, G.: Ubiquitous recommender systems. Computing **96**(3) (2014) 223–257
14. Adomavicius, G., Mobasher, B., Ricci, F., Tuzhilin, A.: Context-aware recommender systems. AI Magazine **32**(3) (2011) 67–80
15. Panniello, U., Tuzhilin, A., Gorgoglione, M.: Comparing context-aware recommender systems in terms of accuracy and diversity: Which contextual modeling, pre-filtering and post-filtering methods perform the best. User Modeling and User-Adapted Interaction **24**(1-2) (2014) 35–65
16. Gorgoglione, M., Panniello, U., Tuzhilin, A.: The effect of context-aware recommendations on customer purchasing behavior and trust. In: Fifth ACM Conference on Recommender Systems (RecSys 2011), ACM (2011) 85–92
17. Agrawal, R., Gollapudi, S., Halverson, A., Ieong, S.: Diversifying search results. In: Second ACM International Conference on Web Search and Data Mining (WSDM'09), ACM (2009) 5–14

Herramienta Colaborativa Multidispositivo para la Edición de Modelos basada en EMF

Miguel A. Teruel, Arturo C. Rodríguez, Elena Navarro, Pascual González

LoUISE Research Group, Universidad de Castilla-La Mancha
{MiguelAngel.Teruel,Arturo.Rodriguez, Elena.Navarro,
Pascual.Gonzalez}@uclm.es

Resumen. En este trabajo se presenta un editor de modelos que permitirá el trabajo colaborativo con modelos pertenecientes a cualquier metamodelo creado con EMF. Además, este editor permite su uso desde una gran variedad de dispositivos, facilitando una colaboración efectiva gracias al awareness que proporciona tanto sobre el propio proceso de edición como de los usuarios que en ella colaboran. Además, la implementación como servicios webs de la de gestión de modelos EMF y del soporte a características de awareness permite su reutilización en futuras aplicaciones.

Palabras clave: editor modelos, colaboración, multidispositivo, EMF, Workspace Awareness, servicio web

1 Introducción

Durante los últimos años, estamos viendo cómo la colaboración entre usuarios está llegando a todo tipo de aplicaciones. Ya sean aplicaciones clásicas, como editores de texto [1] y juegos online [6], o las vanguardistas aplicaciones Post-WIMP [3]. Además de éstas, otras aplicaciones orientadas a dominios más específicos, como la edición de modelos, también están siendo adaptadas para que varios usuarios puedan trabajar colaborativamente [5].

No obstante, los editores colaborativos actuales están enfocados a editar modelos que son instancias de un metamodelo concreto, limitando su aplicación a otros dominios o contextos de interés. Por esa razón, en este trabajo se presenta un editor de modelos colaborativo basado en Eclipse Modeling Framework (EMF) [7]. Gracias a su integración con EMF (transparente al usuario), el editor de modelos permite su adaptación a cualquier metamodelo que haya sido diseñado con dicho framework. Dicha integración se realiza por medio de un servicio web que permite independizar la plataforma y tecnología de la aplicación cliente. Junto a este servicio web se ha implementado otro servicio encargado de dar soporte a la colaboración, proporcionando a los usuarios el *awareness* (o consciencia) necesario, tanto de los otros colaboradores como de los elementos de los modelos que están siendo manipulados.

Este trabajo se estructura de la siguiente manera. Tras esta introducción, en la Sección 2, se presentará el servicio web encargado de la gestión de los modelos EMF. A continuación, la Sección 3 presenta el servicio web que ofrece soporte al awareness.

La aplicación colaborativa cliente que consume dichos servicios se presenta en la Sección 4. Finalmente, la Sección 5 presenta nuestras conclusiones y trabajos futuros.

2 Arquitectura de un Servicio Web basado en EMF

Eclipse Modeling Framework (EMF) [7] es uno de los frameworks de modelado y generación de código más extendidos en el Desarrollo Dirigido por Modelos. Esta tecnología, desarrollada en Java, se integra en el IDE Eclipse y permite crear, editar e instanciar modelos e implementar de forma automática un esqueleto de código a partir de los mismos. Además, a diferencia de otras herramientas de este tipo, EMF está diseñado para ofrecer de forma real un punto de partida para una programación eficiente. La definición del metamodelo que guía el proceso puede hacerse mediante tres métodos: (1) Esquema XML, (2) diagrama de clases UML y (3) interfaces Java.

Una vez definido el metamodelo, EMF genera de forma automática varios componentes que permiten al usuario la instanciación y el manejo de dichos modelos. Entre estos componentes podemos encontrar un plug-in para Eclipse que permite crear instancias del metamodelo a través de un wizard y las clases java que permiten manejar los elementos del modelo a nivel de programación.

Esta forma de trabajar, aunque no carece de una enorme potencia, conlleva dos limitaciones. Por un lado, requiere que el usuario tenga conocimiento acerca del uso de Eclipse. Esto hace que usuarios no ligados de forma directa a la informática encuentren dificultades importantes a la hora de trabajar con la edición de modelos. A fin de solucionar esta limitación, es posible construir software que, usando de base EMF, ofrezca una interfaz adecuada para este tipo de usuarios. Sin embargo, esto nos llevaría a encontrar una segunda limitación, ya que estas aplicaciones standalone sólo pueden ser desarrolladas en Java y son desarrolladas exclusivamente para un metamodelo concreto.

Por ello, hemos desarrollado un servicio web que ofrece una vía de comunicación entre aplicaciones destinadas a trabajar con modelos y el framework EMF (Fig. 1). De esta manera, es posible desarrollar aplicaciones cuya interfaz de usuario y lógica no asociada a la gestión de modelos son totalmente independientes de todo lo relacionado con éstos, permitiendo mejorar así la portabilidad dado que se pueden integrar mediante cualquier tipo de tecnología en esas partes de la aplicación.

De esta manera, sólo el servicio web hace uso de las clases de EMF, actuando sus métodos como una capa intermedia que consumen las peticiones externas, usan el

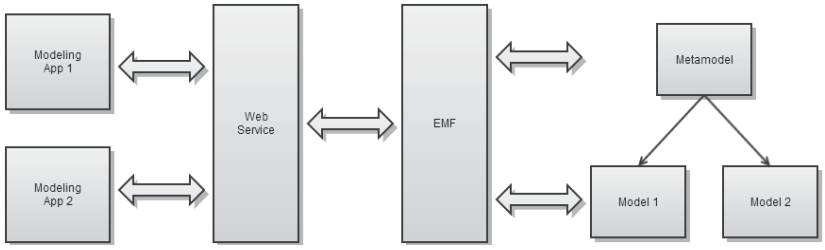


Fig. 1. Arquitectura del servicio web basado en EMF

framework para consultar el modelo en función de la información requerida y devuelven dicha información en formato XML. El servicio web ha sido implementado usando Axis e incluyendo las librerías necesarias del framework EMF en el servidor Apache Tomcat en el que ha sido desplegado. Dicho servicio web se ha desarrollado agrupando los diferentes métodos en dos grupos distintos:

- *Métodos de consulta del metamodelo*: con estos métodos las aplicaciones clientes pueden obtener información sobre las metaclases y las relaciones del metamodelo, saber si una metaclase es abstracta, las cardinalidades de las relaciones, etc.

```
public String getClassInfo(String model,String className,String info)
public String getReferenceInfo(String model,String element,
  String reference,String info)
```

  El primer método devuelve información concreta de una metaclase del modelo. Para ello se pasa al servicio web el nombre del modelo, el nombre de la clase y la información requerida. El segundo método devuelve información acerca de una referencia del metamodelo. En este caso, además del modelo, la referencia y la información requerida, es necesario saber el elemento del que parte la referencia.

- *Métodos de gestión del modelo*: estos métodos permitirán consultar y modificar objetos del modelo o crear nuevas instancias del mismo.

```
public getElement(String model,String type,String id)
public createElement(String model,String parent,String type,String id)
```

  El primer método recupera un objeto del modelo mientras que el segundo crea un nuevo objeto en el modelo. El servicio web se ocupa también de la persistencia del modelo. De esta forma la edición de un modelo determinado no está ligada a un solo usuario, sino que puede hacerse de manera colaborativa.

3 Servicio Web para dar soporte a Workspace Awareness

Paralelamente al desarrollo de la colección de servicios web para EMF, ha sido implementada una nueva biblioteca de métodos web con el objeto de mejorar la interacción colaborativa de los usuarios. Concretamente, este servicio web da soporte a la interpretación *Workspace Awareness* (WA) propuesta por Gutwin [2]. Mediante WA, se pretende dar respuesta a una serie de elementos de los cuales un usuario necesita ser consciente cuando trabaja con un sistema colaborativo. Por ejemplo, WA plantea elementos de consciencia como *quién* hay disponible en el sistema, *qué* están haciendo los demás, *donde* están trabajando, *cuando* realizaron una determinada acción, etc. Concretamente, los elementos que recoge WA pueden verse en la Tabla 1.

Tabla 1. Elementos de Workspace Awareness

Tiempo	Categoría	Elemento	Preguntas específicas
Presente	Quién	Presencia Identidad Autoría	¿Hay alguien en el espacio de trabajo? ¿Quién está participando? ¿Quién está haciendo eso?
	Qué	Acción Intención Artefacto	¿Qué están haciendo? ¿Con que objetivo realiza esa acción? ¿Con que objeto están trabando?
	Dónde	Localización Mirada Vista Alcance	¿Dónde están trabajando? ¿Dónde están mirando? ¿Qué pueden ver? ¿Qué pueden alcanzar?
Pasado	Cómo	Histórico de acciones Histórico de artefactos	¿Cómo ocurrió eso? ¿Cómo llegó el artefacto a ese estado?
	Cuándo	Histórico de eventos	¿Cuándo ocurrió ese evento?
	Quién	Histórico de presencia	¿Quién estuvo aquí y cuándo?
	Dónde	Histórico de localización	¿Dónde ha estado esa persona?
	Qué	Histórico de acción	¿Qué ha estado haciendo esa persona?

Los elementos anteriormente mostrados han sido implementados como un servicio web Axis2 con motor de BBDD MySQL. Al igual que ocurría con el servicio de edición de modelos, la utilización de tecnología de servicios web permite implementar aplicaciones que necesiten soporte de WA utilizando cualquier tecnología. A continuación, a modo de ejemplo, se muestran los métodos correspondientes a los dos primeros de los elementos de WA relacionados con el presente (*quién – presencia* y *quién - identidad*):

```
public bool isAnyone()
public Participant[] whoIsParticipating()
public void addParticipant(Participant p)
public void removeParticipant(Participant p)
```

El primero de estos cuatro métodos web (isAnyone) proporciona información acerca de si hay alguien disponible en el sistema con quien colaborar, mientras que el segundo (whoIsParticipating) informa acerca de quiénes son esos participantes. Para el correcto funcionamiento de estos métodos web, otros dos métodos auxiliares han sido implementados (addParticipant y removeParticipant). El primero de estos dos métodos se utiliza para comunicar al servidor que un participante se ha conectado al sistema, mientras que el segundo permite eliminar a un participante.

4 Cliente multidispositivo para la edición de Modelos

Una vez se han definido los servicios web que permiten la edición de modelos con soporte a los diferentes elementos de WA, se presenta a continuación la aplicación cliente creada para su utilización. Concretamente, dos aplicaciones clientes usuario distintas han sido implementadas. La primera de ellas se basa en una interfaz web pensada tanto para dispositivos de escritorio como para móviles, en la que se centra

Fig. 2. Editor de modelos ejecutado sobre distintos dispositivos

este trabajo, mientras que la segunda está pensada para ser usada por hasta cuatro usuarios simultáneamente en una mesa Microsoft PixelSense.

Este editor de modelos ha sido implementado como aplicación web con el fin de que pueda ser soportado por una gran variedad de dispositivos (Fig. 2). Concretamente, fue implementando usando una combinación de tecnologías web como ASP.net y JavaScript, las cuales han permitido una total integración con los dos servicios web desarrollados, así como dotando a la aplicación de compatibilidad con cualquier dispositivo que soporte HTML5.

Gracias a esta aplicación, varios usuarios pueden trabajar de forma colaborativa sobre un mismo modelo instancia de un metamodelo EMF. La aplicación cliente accede al modelo mediante el servicio web detallado en la sección 2. Esto permite la separación de la interfaz de la aplicación del metamodelo concreto empleado, haciendo así la aplicación compatible con cualquier metamodelo diseñado con EMF. Además, es importante resaltar que dicha aplicación se ha desarrollado utilizando las heurísticas descritas en un trabajo previo [4]. Estas heurísticas se han definido con dos objetivos distintos: (i) identificar metáforas que asocian tipos de gestos con tareas concretas de edición; (ii) permitir, gracias a dichas metáforas, que personas sin conocimientos de edición de modelos puedan participar en dicho proceso.

Finalmente, esta aplicación utiliza el servicio web descrito en la sección 3, para permitir a los usuarios trabajar colaborativamente proporcionándoles el awareness necesario tanto de los usuarios como de los elementos del modelo que estén editando. De hecho, directamente ligadas al awareness, la aplicación dispone de las siguientes capacidades relacionadas con el WA descrito por Gutwin (ver Tabla 1):

- *Usuarios conectados*: Muestra una lista de los usuarios conectados (*quién - identidad*). Además, el icono del menú de usuario cambia en función de si hay alguien más en el sistema, mostrando uno o varios usuarios (*quién - presencia*).
- *Historial de usuarios*: Mediante el menú de usuario se puedo obtener información acerca qué está haciendo un usuario (*qué - acción*), así como obtener una lista de las acciones pasadas (*qué - histórico de acción*).

- *Historial de elementos*: Al igual que ocurre con los usuarios, pueden consultarse todos los cambios que han acontecido sobre un elemento de un modelo (*cómo – histórico de artefacto*).

5 Conclusiones y Trabajos Futuros

En este trabajo se presenta una propuesta la edición de modelos colaborativa multi-dispositivo. Para dar a dicha edición de modelos colaborativa, dos servicios web (que pueden ser reutilizados por otras aplicaciones) han sido desarrollados. Uno de dichos servicios permite la edición de modelos EMF de forma independientemente de la plataforma o tecnología utilizada. El otro servicio desarrollado soporte la interpretación de WA definida por Gutwin para permitir el trabajo colaborativo.

Paralelamente a esta aplicación multidispositivo, se está desarrollando una segunda que se ejecutar sobre Microsoft PixelSense, haciendo uso de los servicios web descritos. De esta manera, las dos aplicaciones podrán comunicarse, permitiendo a los usuarios de ambas trabajar de forma colaborativa sobre un mismo modelo.

Agradecimientos. Este trabajo ha sido financiado por el Ministerio de Economía y Competitividad, así como por los fondos FEDER de la Unión Europea mediante el proyecto inspire (TIN2012-34003). Además, ha sido financiado por el Ministerio de Educación, Cultura y Deporte mediante una beca FPU (AP2010-0259). Nos gustaría agradecer el trabajo desarrollado por Almudena Alcaide Sánchez y Miguel Saiz Sánchez en el transcurso de su Trabajo Fin de Grado y que ha ayudado a hacer realidad las ideas aquí presentadas.

Referencias

1. Dekeyser, S., Watson, R.: Extending Google Docs to Collaborate on Research Papers. , Toowoomba, Australia (2006).
2. Gutwin, C., Greenberg, S.: A Descriptive Framework of Workspace Awareness for Real-Time Groupware. Comput. Support. Coop. Work 11 (3), 411–446 (2002).
3. Jetter, H.-C., Zöllner, M., Gerken, J., Reiterer, H.: Design and Implementation of Post-WIMP Distributed User Interfaces with ZOIL. Int. J. Hum. Comput. Interact. 28 (11), 737–747 (2012).
4. Lopez-Jaquero, V., Navarro, E., Montero, F., González, P.: Metamodels Infrastructure and Heuristics for Metamodel-Driven Multi-touch Interaction. 14th IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT 2013). pp. 210–227, Springer, Cape Town, South Africa (2013).
5. Mougenot, A., Blanc, X., Gervais, M.-P.: D-Praxis : A Peer-to-Peer Collaborative Model Editing Framework. 9th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS'09). pp. 16–29, , Florence, Italy (2009).
6. Nardi, B., Harris, J.: Strangers and friends: collaborative play in world of warcraft. 20th anniversary conference on Computer supported cooperative work (CSCW'06). p. 149, ACM Press, Banff, Canada (2006).
7. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional (2008).

A CSCW Requirements Engineering CASE Tool: Development and Usability Evaluation*

Miguel A. Teruel, Elena Navarro, Víctor López-Jaquero, Francisco Montero, Pascual González

LoUISE, Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, Avda. España s/n, 02071 - Albacete
{miguel, enavarro, victor, fmontero, pgonzalez}@dsi.uclm.es

Abstract. An experiment was performed in order to assess the usability of CSRML Tool 2012 (CT'12), a tool to specify the requirements of CSCW systems. The effectiveness and efficiency of CT'12 were evaluated by means of several experimental tasks. Moreover, the user's satisfaction was evaluated by using both a classical survey and analyzing the participants' facial expressions when performing the experiments. Furthermore, this evaluation has been reported by using the ISO/IEC 25062:2006 [2], thus making its results comparable with other usability assessments which follow this international standard. Finally, details about how to develop a CASE toll for a Domain Specific Language was provided in a tutorial style.

Palabras clave: Usability evaluation; CASE tool; CSRML; Requirements engineering; CSCW; ISO/IEC 25062:2006

1 Context of the proposal: RE CASE Tools for CSCW Systems

Part of the success of a Requirement Engineering (RE) technique lies in its supporting tool, with which to specify and validate requirements models. As happens with classical RE techniques, CSRML (Collaborative Systems Requirements Modeling Language) [7] needed a CASE tool to help Requirement Engineers specify requirements for CSCW systems [4]. With this aim CSRML Tool 2012 (CT'12) [6] was developed as a Visual Studio extension to properly specify and validate CSRML graphical requirements models, thus avoiding potential sources of specification mistakes.

2 Usability Evaluation

28 fourth-course Computer Science students took part in this evaluation. They were asked to carry out a series of modifications to an incomplete CSRML requirements specification. In ensure that the participants were familiar with the domain, the pro-

* Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, González, P.: A CSCW Requirements Engineering CASE Tool: Development and Usability Evaluation. Information and Software Technology, 56(8): 922-949. DOI: 10.1016/j.infsof.2014.02.009

vided CSRML requirements specification corresponded to a multi-user online strategy game, well known by the participants due to their studies and age. Usability was assessed by measuring the task's completion rate, the elapsed time, number of accesses to the help system of the tool and the instructor's verbal assistance. Globally, these results were considerably positive, with an 89.29% of the tasks correctly completed, with a mean completion time about 31 minutes (being 19 minutes the time that a CSRML reference expert took to complete the tasks) and an average efficiency of 17.85 correct tasks per hour. However, there was a surprising fact: the a priori easiest task obtained the worst completion result, making evident a problem with one on the CSRML elements that was solved afterwards in CT'13 [5].

As far as user's satisfaction is concerned, the used facial expression matching technique [1] showed up that when the participants dealt with easy tasks, they tended to behave pleasantly, but as the difficulty increased, this tendency changed to a calmer and less pleasant behavior, probably because they started getting used to interacting with CT'12 during the medium tasks. Nevertheless, when they finally had to deal with the hard tasks, there was a drastic change to more unpleasant behavior, due to the high complexity. These results confirm the pre-assigned task difficulty levels used in our study. This means that the tasks that we considered easy were in fact easy for the participants (as were the medium and hard). Regarding the questionnaire, the results obtained were satisfactory, with an average result for the whole questionnaire [3] of 6.06, taking into account that the questions were answered in a scale from 1 (strongly disagree) to 7 (strongly agree).

Acknowledgments. This work has been funded by the Spanish Ministry of Economy and Competitiveness and by the FEDER funds of the EU under the project Grant insPIre (TIN2012-34003). It has also been funded by Spanish Ministry of Education, Culture and Sport thanks to the FPU scholarship (AP2010-0259).

References

1. Desmet, P.M.A.: Measuring Emotions: Development and application of an instrument to measure emotional responses to products. In: Blythe, M.A. et al. (eds.) Funology: From Usability to Enjoyment. pp. 1–13, Springer (2003).
2. ISO/IEC 25062: Software engineering—Software product Quality Requirements and Evaluation (SQuaRE)—Common Industry Format (CIF) for usability test reports. (2006).
3. Lund, A.M.: Measuring Usability with the USE Questionnaire. STC Usability SIG Newsl. 8 (2), (2001).
4. Schmidt, K., Bannon, L.: Taking CSCW seriously. Comput. Support. Coop. Work 1 (1-2), 7–40 (1992).
5. Teruel, M.A.: CSRML Tool 2013, http://bit.ly/csrml13, (2013).
6. Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., González, P.: CSRML Tool: a Visual Studio Extension for Modeling CSCW Requirement. 6th International i* Workshop (iStar'13). pp. 122–124, CEUR Vol-978, Valencia (Spain) (2013).
7. Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F., González, P.: CSRML: A Goal-Oriented Approach to Model Requirements for Collaborative Systems. In: Jeusfeld, M. et al. (eds.) 30th International Conference on Conceptual Modeling (ER'11). pp. 33–46, Springer Berlin Heidelberg, Brusells, Belgium (2011).

WebMakeup: An End-user Tool for Web Page Customization

Oscar Díaz[1], Cristóbal Arellano[1], Iñigo Aldalur[1], Haritz Medina[1], Sergio Firmenich[2]

[1] University of the Basque Country (UPV/EHU), San Sebastián (Spain)
(oscar.diaz, cristobal.arellano, inigo.aldalur)@ehu.es,
hmedina002@ikasle.ehu.es,
[2] LIFIA, Facultad de Informática (Argentina)
sergio.firmenich@lifia.info.unlp.edu.ar

Abstract. The Personal Web refers to a collection of technologies that confer the ability to reorganize, configure and manage online content rather than just viewing it. The main forms of Web content are HTML pages. HTML pages are represented as DOM trees, hence the Web is conceived as a "forest of DOM trees". The vision is for users to "prune" (removing nodes) or "graft" (adding nodes) existing DOM trees to improve their Web experience. Hence, Web content is no longer consumed as canned by Web masters. Rather, users can remove content of no interest to them, or placing new content from somewhere else. This vision accounts for a post-production user-driven Web customization. Being user driven, appropriate abstractions and tools are needed. The paper introduces an IDE (realized as a plugin from Chrome) to empower non-programmers to achieve HTML rearrangement.

Keywords: Web Modding, Web Widget, End User Programming, Visual Programming, Domain Specific Languages, WebMakeUp

1 Motivation & Background

The Personal Web refers to "a collection of technologies that confer the ability to reorganize, configure and manage online content rather than just viewing it" [1]. The Personal Web can be seen as a continuation of the Web2.0 movement. While Web2.0 puts *content authoring* in the user's hand [2], the Personal Web accounts for *content rearrangement* at users' will. Web content can be delivered through different formats and means (e.g. RSS feeds, open APIs) and hence, content rearrangement can take different forms (e.g. RSS readers, Web mashups). However, the primary format of Web delivery is HTML. *This is the focus of WebMakeup. WebMakeup* is a browser plugin that empowers user to re-arrange Web content at wish. Web content is no longer consumed as canned by Web masters. Rather, users can remove content of no interest to them (leading to less cluttered pages and reducing scrolling) or placing new content from somewhere else (reducing moving back and forth between sites, and providing a single viewing

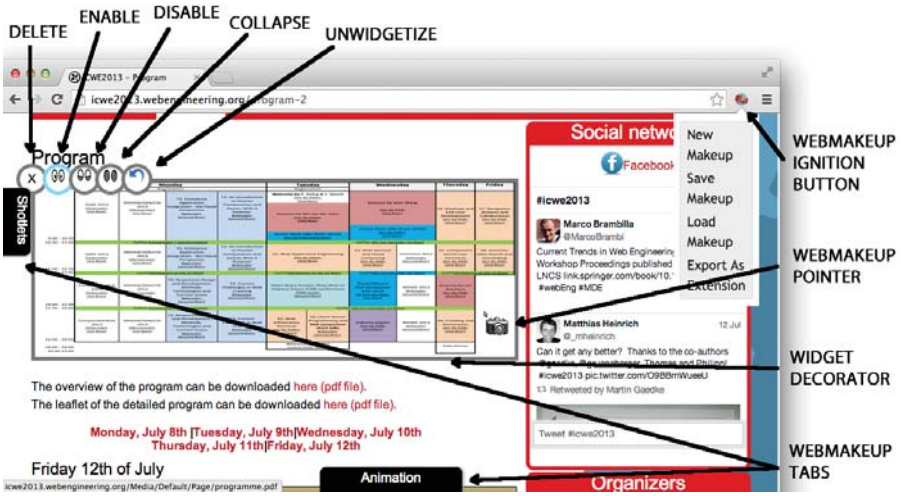


Fig. 1. WebMakeup: Widgetization.

context). This vision accounts for a post-production user-driven Web customization. Next, we introduce *WebMakeup* through an example. A video is available at `http://onekin.org/downloads/public/WebMakeup/videoJISBD.mov`.

2 A session with *WebMakeup*

Scenarios that benefit from the Personal Web include websites that are frequently accessed but designed for a general audience. This might introduce a penalty on the user experience for more situational, idiosyncratic usages. Conference websites are a case in point. We take the sample case of the *ICWE'13* website, specifically the page `http://icwe2013.webengineering.org/program-2`. This page neatly presents the conference program for this conference. As most conference sites, the page includes an exhaustive list of all sessions and papers to be presented together with some decorator information such as sponsors and tweets. An eye-catching design aims at capturing the attention of potential attendees indicating the surfeit of talks they can enjoy if they attend the conference. However, this attractive design might not match the needs of the actual attendees during the conference. Their aims are different. In case of parallel sessions, they might already decide which sessions to attend so that displaying the full program forces scrolling and reduces focus. The venue, dates and organization buttons no longer make sense once you are already at the venue. Likewise, eye-catching effects do no longer serve their purpose (having the attention of *potential attendees*) but become distracting for *actual attendees*. In addition, actual attendees might like to know the weather forecast at the conference place, and some might also like to know the *Google Scholar* citation indexes of their colleagues. This

will force attendees to move to a different browser tab. That said, the discomfort is not so important to pay off for a website reorganization. But, what if users could customize this page on their own? This is the aim of *WebMakeup*.

WebMakeup is a plugin for the *Google Chrome* browser. Its installation is reflected through the *WebMakeup* button at the right of the address bar (see Figure 1). On clicking this button, the current page is turned into "an augmentation canvas", i.e. two side tabs pop up (i.e. *widget tab* and *animation tab*), the pointer is turned into a camera, and a grid-like structure is interspersed on top of the current DOM tree. The editor is ready for specifying the augmentation. Next, we describe the different operations in the order we feel natural to use, though *WebMakeup* does not impose any order.

Widgetization (Figure 1). As the user moves the cursor around the screen, the DOM node under the current cursor location is highlighted. By clicking, the user identifies a "unit of rendering" and the DOM node is turned into a widget clone. This implies enhancing the DOM node with the widget decorator. This decorator exhibits the different widget states and GUI events the widget can respond to. At this point, the user can opt for supressing the clone (i.e. the X option on the decorator) or parameterizing it. This is achieved by double clicking. This flips over the clone to show up the parameters. Click again and the clone turns to its rendering position. The case study provides different widgetization examples. The heading conference logo, *Aalborg* passing pictures and the program at a glance are all removed.

Mining. At any time, the user can open other browser tabs, request new pages and mine for widgets. The latter is achieved by extending the right-click contextual menu with the *mineIt* item. On selecting it, a grid-like structure is interspersed on top of the current DOM tree. As the user moves the cursor around the screen, the DOM node under the current cursor location is highlighted. By clicking, the user enacts *mine()* upon this DOM node. This function digs into the HTML code to determine whether cloning or cropping is the most appropriate technique. In addition, it prompts for a label to identify the so-obtained widgets. Mined widgets are kept locally as part of the browser state for self-consumption. Widgets are available under the *shotlet tab*. For the case study, two widgets have been extracted: *weatherForecast* from `http://www.zoover.co.uk/denmark/jutland/aalborg/weather` and *gScholarIndices* from `http://scholar.google.com/`.

Cloning (Figure 2). By clicking on the *shotlet tab*, mined widgets are displayed. Just drag&drop for widget cloning. Placement heuristics will warn or prevent from dropping the clone in certain places. Double clicking turns down the clone to show its parameters. At this time, binding might occur. Besides constants, users can bind parameters to values to be extracted from the hosting page at runtime. The former is illustrated by the *weatherForecast* clone where the location parameter is set to *Aalborg*. By contrast, *gScholarIndices* bounds its scholar name parameter to the author of the last tweet. At run time, changes on the *Twitter* box will cause an automatic refreshment of *gScholarIndices*. Dynamic binding is supported through XPath expressions. At parameterization

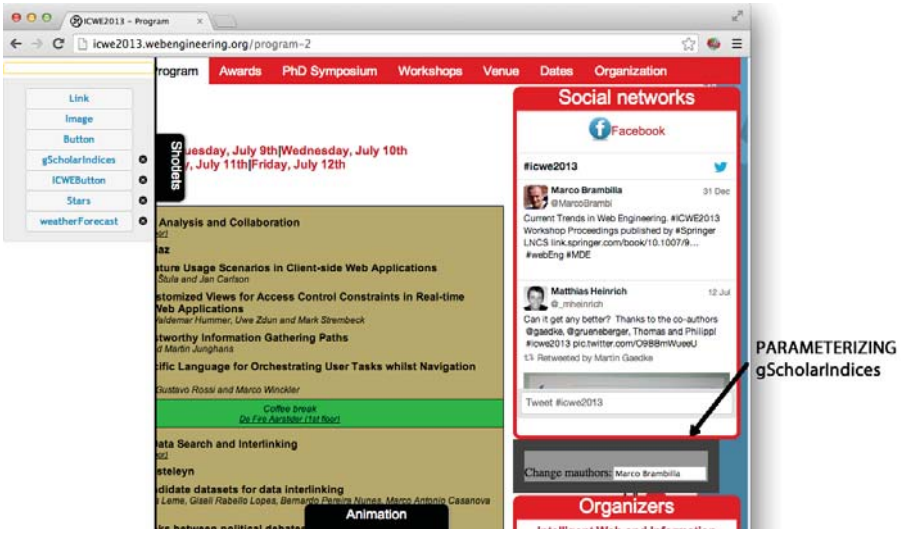


Fig. 2. WebMakeup: Cloning.

time, users can just select the desired value from the underlying page, and paste it into the parameter input box. *WebMakeup* will infer the XPath expression and attach it to the desired parameter.

The above scenarios can be intertwined at wish. If interrupted, laymen can save partial results for later resumption (see Figure 1, ignition-button drop-down menu). On completion, the WA can be exported as a *Google Chrome* extension. For our running example, the generated extension is available at `http://onekin.org/downloads/public/WebMakeup/extensionJISBD.zip`. Once the extension is installed, the augmentation will take place automatically.

Acknowledgments.

This work is co-supported by the Spanish Ministry of Education, and the European Social Fund under contract TIN2011-23839. Aldalur has a doctoral grant from the Spanish Ministry of Science & Education.

References

1. L. Johnson, A. Levine, and R. Smith. *The 2009 Horizon Report*, chapter Two to Three Years: The Personal Web, pages 19–22. 2009.
2. T. O'Reilly. What Is Web 2.0, 2005. http://oreilly.com/web2/archive/what-is-web-20.html.

Metodología para Diseñar, Desarrollar y Evaluar una Plataforma de Entrenamiento en Desarrollo Global de Software

Miguel J. Monasor[a,b], Aurora Vizcaíno[a], Mario Piattini[a],
John Noll[b], Sarah Beecham[b]

[a] Grupo de Investigación Alarcos, Instituto de Tecnologías y Sistemas de Información, Escuela Superior de Informática, Universidad de Castilla-La Mancha, 13071, Ciudad Real, España
MiguelJ.Monasor@gmail.com, {Aurora.Vizcaino, Mario.Piattini}@uclm.es

[b] Lero, The Irish Software Engineering Research Centre, Department of Computer Science and Information Systems, University of Limerick, Limerick, Ireland
{Sarah.Beecham, John.Noll}@lero.ie

Resumen. La formación en Desarrollo Global de Software (DGS) ha cobrado especial relevancia en lo últimos años, tanto en el ámbito académico como en el de la industria. Las empresas ofrecen formación en DGS en áreas como el entrenamiento cultural y lingüístico, liderazgo, negociación o habilidades comunicativas. Por otro lado, las universidades también ofrecen formación en DGS aunque con objetivos más genéricos. Considerar las necesidades de ambos entornos (industria y academia) es un factor clave para proporcionar una solución formativa efectiva. VENTURE es un entorno basado en simulación para proporcionar entrenamiento en interacciones textuales de DGS. Este entorno considera diferentes tipos de problemas del DGS incluyendo barreras comunicativas tanto lingüísticas como culturales. Este artículo presenta la metodología utilizada para diseñar, desarrollar y evaluar VENTURE. Una vez descrita la metodología nos centraremos en explicar en detalle las últimas etapas de la misma que fueron: la evaluación de viabilidad, el estudio de expertos, el estudio de campo y el estudio comercial.

Palabras clave: Desarrollo Global de Software; Educación; Docencia; Formación; Equipos Virtuales; Simulación; Metodología; Evaluación

1 Introducción

El Desarrollo Global de Software (DGS) implica diversos problemas relacionados con la comunicación, coordinación y colaboración de los diferentes participantes [1]. Las distancias geográficas y temporales conllevan diferencias culturales y lingüísticas que complican las interacciones entre los miembros del equipo [2]. Algunos de los conflictos están relacionados con la falta de confianza, liderazgo o problemas de comunicación.

Con el objetivo de preparar a los ingenieros para enfrentarse a los desafíos del DGS es esencial una adecuada formación tanto en ámbitos académicos como en la industria [3].

Revisando la literatura descubrimos que algunas universidades proporcionan entrenamiento específico en DGS. Una de las estrategias empleadas consiste en colaborar con otras universidades de diferentes países para desarrollar proyectos conjuntos en los que los estudiantes interaccionan con equipos de estos países reproduciendo escenarios de entornos globales de desarrollo [4], [5]. Por otro lado, las iniciativas de la industria consisten generalmente en aprovechar el conocimiento de sus expertos para proporcionar entrenamiento concreto en base a sus experiencias [6]. Las multinacionales suelen invertir en entrenar a sus miembros en destrezas relacionadas con el DGS tales como el liderazgo, asertividad, negociación, diferencias culturales y diferencias lingüísticas [7]. Sin embargo, los métodos de entrenamiento actuales presentan diversos inconvenientes tales como la necesidad de coordinación del trabajo, dificultad de la evaluación de las destrezas desarrolladas, falta de disponibilidad de recursos o de rigor del entrenamiento.

En este artículo se presenta una solución a dichos inconvenientes: VENTURE (Virtual ENvironmenT for commUnication and collaboRativE training) [8] es un entorno de entrenamiento virtual que simula interacciones textuales entre personas involucradas en equipos virtuales. De esta forma, VENTURE mejora la efectividad de la comunicación entre personas de diferentes culturas y lenguajes en entornos de DGS. Los estudiantes interaccionan con Agentes Virtuales (simulados por avatares) que interpretan un papel en la interacción simulada y en la que pueden participar de manera independiente.

En este artículo presentamos la metodología que se ha usado para diseñar, desarrollar y evaluar VENTURE de manera iterativa. La metodología consta de las siguientes etapas:

1. Revisión Sistemática de la Literatura en DGS para descubrir las áreas en las que centrar la investigación.
2. Revisión Sistemática de la Literatura en formación en DGS para estudiar los métodos formativos actuales, sus ventajas e inconvenientes.
3. Diseño de un entorno de entrenamiento que pueda simular interacciones textuales que reproduzcan escenarios realistas de DGS.
4. Evaluar la viabilidad y adecuación del entorno a través de una evaluación de viabilidad en la que se analizó la opinión de expertos.
5. Implementar el entorno y llevar a cabo una evaluación de expertos sobre dicha implementación con el objetivo de estudiar aspectos técnicos y de contenido.
6. Mejorar el entorno y llevar a cabo un estudio de campo para probar que el entorno es efectivo en sus objetivos formativos.
7. Llevar a cabo un estudio comercial para evaluar las opciones comerciales del entorno en base a una metodología orientada al negocio.
8. Definir objetivos futuros para satisfacer los requisitos comerciales y mejorar la eficiencia formativa del entorno.

Este trabajo comenzó como un proyecto inicialmente orientado al ámbito académico, sin embargo, dado que los problemas en DGS están presentes en un importante sector de la industria, se decidió llevar a cabo un estudio comercial que

pudiera evaluar las posibilidades de VENTURE. Dicho estudio incluyó entrevistas semiestructuradas con expertos de empresas multinacionales que sirvieron para identificar las necesidades reales y los requisitos para llevar a cabo la comercialización. Estas reuniones ayudaron a definir áreas en las que centrar el trabajo futuro basado en necesidades tanto académicas como de mercado.

2 VENTURE

VENTURE [8], [9] permite simular interacciones textuales de DGS entre estudiantes y Agentes Virtuales. Los Agentes Virtuales interaccionan de manera autónoma simulando formar parte de un equipo virtual. Los escenarios quedan disponibles a los estudiantes a través de un portal *e-learning* que también contiene material teórico.

Los estudiantes son evaluados de manera automática durante el transcurso de las simulaciones. De igual forma se pueden considerar evaluaciones manuales que los instructores pueden usar para considerar objetivos específicos.

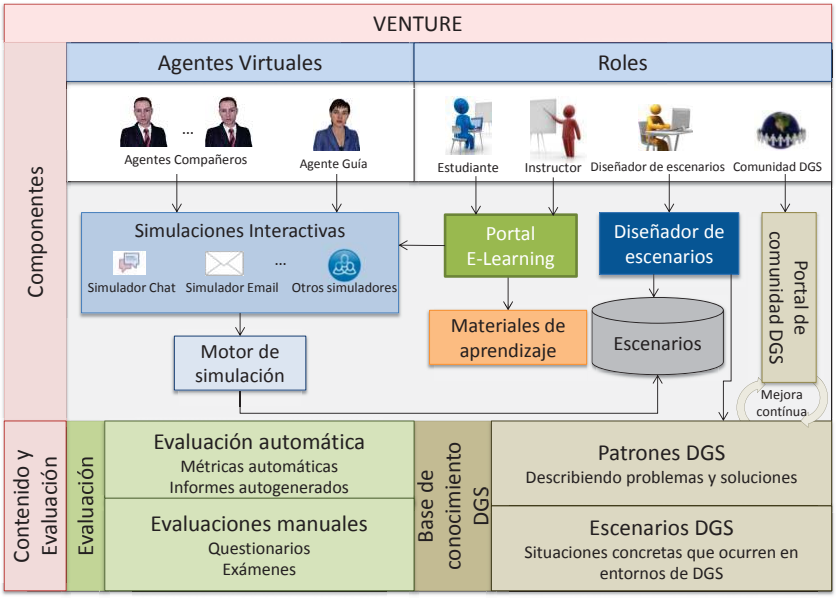


Fig 1. VENTURE

La herramienta tiene un diseñador de escenarios que facilita la creación y modificación de las simulaciones, permitiendo definir los Agentes Virtuales y los problemas de DGS que los estudiantes afrontarán durante la simulación. Sin embargo, una de las dificultades de diseñar escenarios de DGS es tener acceso a problemas y situaciones reales para poder reproducirlas. Para solventar esto, el entorno incorpora una plataforma colaborativa que facilita que la comunidad de DGS pueda colaborar con escenarios y patrones de DGS. De esta forma, se facilita

la adquisición de material que puede emplearse para la generación de escenarios de entrenamiento específicos de DGS relativos a captura de requisitos, diseño, implementación, pruebas, etc.

VENTURE hace uso de dos tipos de Agentes Virtuales (ver Figura 1): Agentes Compañeros, que juegan un rol en las interacciones textuales de DGS y el Agente Guía, que indica a los estudiantes cómo deben interaccionar y les corrige en caso necesario.

Las interacciones simuladas se pueden llevar a cabo a través de un simulador de chat y a través de un simulador de e-mails:

A. **Simulador de Chat**: permite entrenar interacciones textuales síncronas en las que el objetivo del estudiante es obtener información del Agente Compañero dentro del tiempo límite de la simulación. El Agente Guía proporciona correcciones instantáneas y guía al usuario. El Agente Compañero puede, además de comunicarse textualmente, gesticular y hablar imitando el acento de su nacionalidad.

B. **Simulador de E-mails**: Los estudiantes pueden usar el simulador de interacciones textuales asíncronas para intercambiar e-mails con Agentes Virtuales. La Figura 2 muestra un ejemplo de una simulación de email en la que el Agente Compañero (Raúl) comienza la interacción pidiendo a un estudiante de Reino Unido (Sue) que le envíe un documento.

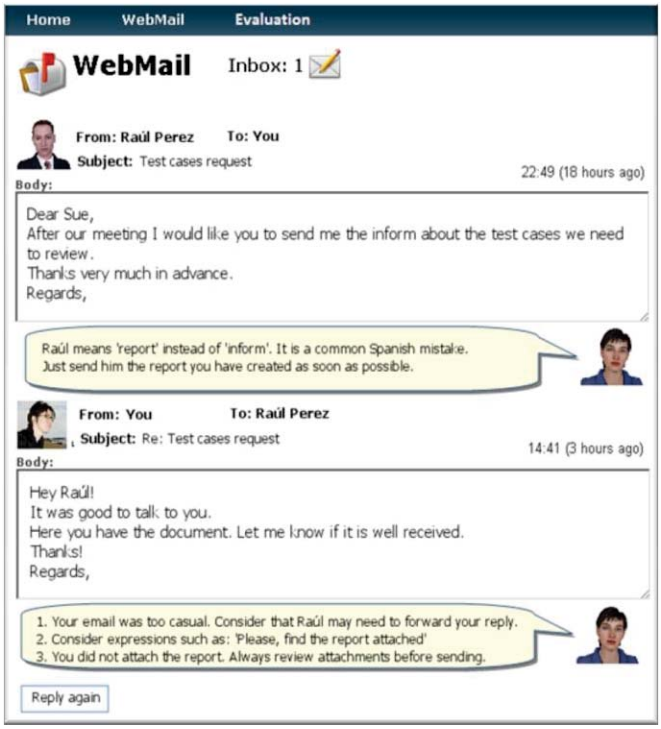


Fig 2. Simulador de e-mails

Se asume que en una conversación por chat anteriormente simulada, Sue y Raúl han hablado sobre un conjunto de casos de prueba y Sue tiene un documento disponible. En esta interacción, Raúl ha cometido un error típico de un español hablando en inglés. El Agente Guía explica inmediatamente dicho error a Sue. Una vez Sue responde, el Agente Guía detecta que ésta ha cometido tres errores y se los explica a Sue para que pueda responder nuevamente tras realizar las correcciones oportunas.

Durante la ejecución de los escenarios, los estudiantes son evaluados de manera automática, teniendo en cuenta reglas lingüísticas, culturales y propias del DGS. De esta forma, tras la finalización de la simulación, pueden visualizar un informe que les indica su rendimiento. Dicho informe incluye un resultado final y resultados independientes para cada una de las destrezas de DGS entrenadas durante la simulación. El diseñador de escenarios permite el diseño de cualquier tipo de interacción textual involucrando diferentes culturas y lenguajes. Una descripción más detallada de VENTURE se puede encontrar en http://global.lero.ie/venture.

3 Metodología

Una vez explicado VENTURE se va a describir la metodología de investigación aplicada para desarrollarlo la cual se basa en una serie de pasos incrementales a través de los que se definió y mejoró el entorno a través de varias evaluaciones [10]. La Figura 3 muestra de forma esquemática dicha metodología que consta de las siguientes fases: El primer paso (fase 1) consistió en estudiar el dominio del DGS a través de una Revisión Sistemática de la Literatura [1] que daba respuesta a la siguiente pregunta de investigación: *¿Qué iniciativas existen para la mejora de procesos de DGS?* Esto sirvió para descubrir los problemas del DGS en los que se debería centrar la investigación. Dicho estudio condujo a la decisión de concentrar los esfuerzos hacia la formación en DGS. La fase 2 de esta investigación consistió por tanto en llevar a cabo una nueva Revisión Sistemática de la Literatura [3]; esta vez en el campo de la formación en DGS, guiada por la siguiente pregunta de investigación: *¿Cuáles son las iniciativas formativas existentes en DGS?*

Como resultado de dicho estudio, se encontraron una variedad de problemas en los métodos tradicionales de entrenamiento, principalmente relacionados con los problemas para coordinar este tipo de formación así como la dificultad para reproducir problemas reales. Esto condujo a una nueva pregunta de investigación: *¿Es factible aplicar métodos efectivos para la formación y entrenamiento del DGS que eviten los problemas de los métodos tradicionales?* Como resultado de las anteriores Revisiones Sistemáticas de la Literatura, se diseñó un prototipo de VENTURE para entrenar DGS basado en simulación [8]. Previo a su implementación definitiva, se llevó a cabo una evaluación de viabilidad en la que expertos proporcionaron sus opiniones y sugerencias acerca de dicho prototipos [11] (fase 3). Tras completar este estudio, la definición del prototipo se mejoró y se implementó en forma de una serie de herramientas.

La fase 4 del método de investigación consistió en llevar a cabo una evaluación de expertos [12], en la que los participantes usaron los componentes de la primera versión del entorno (VENTURE V1) y proporcionaron ideas para mejorar la

efectividad y usabilidad. Esta evaluación se realizó sobre una corta simulación que los participantes ejecutaron.

Tras considerar las sugerencias analizadas en esta fase, se desarrolló una versión mejorada del entorno (VENTURE V2). El último paso [13] (fase 5) se centró en dos objetivos: Primero se condujo un estudio de campo con el objetivo de determinar si los participantes podían efectivamente aprender DGS usando el entorno. Para ello, se desarrollaron dos escenarios de entrenamiento que los participantes ejecutaron. Posteriormente, se llevó a cabo un estudio comercial para investigar las opciones comerciales de VENTURE. Estos dos pasos sirvieron para determinar el trabajo futuro que permita mejorar la eficacia del entorno y adaptarlo a las necesidades de los potenciales clientes.

Consideramos que el método de investigación más apropiado para llevar a cabo esta investigación era *mixed methods*, en el que se combinan métodos tanto cualitativos como cuantitativos [14]. Una de las ventajas de usar *mixed methods* es que los investigadores pueden ver los problemas desde diversos puntos de vista [15].

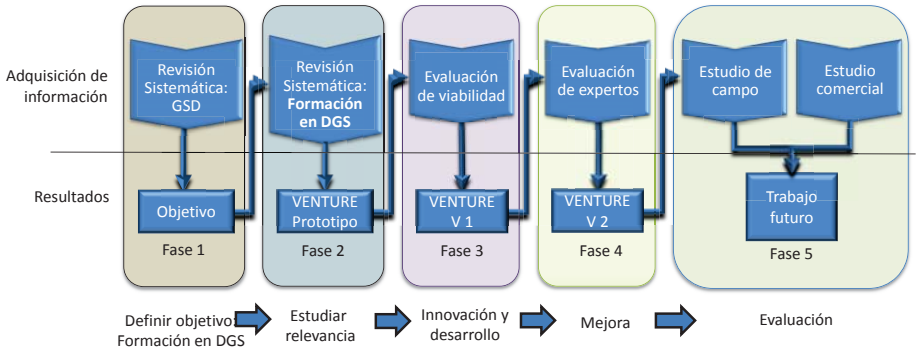


Fig 3. Metodología de investigación

Las siguientes secciones describen la evaluación de viabilidad, el estudio de expertos, el estudio de campo y el estudio comercial.

4 Evaluación de viabilidad

Llevar a cabo una evaluación de viabilidad durante la fase 3 de la metodología, ayudó a determinar cómo mejorar el entorno en una etapa temprana de diseño [11]. El proceso consistió en explicar a los participantes la arquitectura y el propósito de VENTURE, así como su modo de funcionamiento. Los participantes visualizaron una serie de capturas de pantalla del simulador de chat y diseñador de escenarios. Tras esto, completaron un cuestionario con preguntas orientadas a resolver nuestras preguntas de investigación. Cuatro profesionales de diferentes empresas multinacionales (incluyendo dos jefes de proyecto y dos desarrolladores con experiencia de entre 3 a 8 años en DGS) y ocho investigadores, participaron en el cuestionario.

Los resultados mostraron que VENTURE tenía potencial para ser útil y efectivo entrenando procesos de DGS. Tanto investigadores como profesionales de empresas consideraron que VENTURE podría aplicarse en universidades y empresas para proporcionar entrenamiento en áreas de DGS. Los participantes confirmaron que el tipo de entrenamiento que el entorno proporciona puede ayudar a enfocar el entrenamiento de DGS en objetivos concretos que puedan ser reproducidos de una manera sistemática.

Considerar investigadores y profesionales en este estudio fue útil para obtener ideas desde diferentes puntos de vista. Por ejemplo, a los investigadores en general, les agradó la idea de que el entorno pueda proporcionar entrenamiento independiente, adaptable y centrado en problemas concretos que se podría llevar a cabo en cualquier momento sin necesidad de asistir a una clase formal. Puesto que el entorno simula situaciones reales, los investigadores encontraron que la experiencia es motivante e instructiva. Por otro lado, los profesionales de empresas tendieron a considerar un punto de vista más pragmático centrándose en el tiempo y los recursos que podrían ahorrarse aplicando este tipo de entrenamiento. Tanto la flexibilidad para reproducir una gran variedad de escenarios como la independencia del entrenamiento se consideraron de gran valor. Alguno de los participantes incluso sugirió que VENTURE podría emplearse en otros campos de la Ingeniería del Software y no solo en DGS.

Ambos grupos estaban de acuerdo en que el uso del simulador de chat es similar a cualquier otra aplicación de chat. Sin embargo algunos participantes apuntaron que interaccionar con Agentes Virtuales podría tener limitaciones, puesto que los estudiantes no van a interaccionar de la misma forma que si lo hicieran con personas reales, aunque el entorno es perfectamente válido para objetivos de entrenamiento.

Entre los principales puntos débiles detectados por los participantes, se destaca la dificultad de proporcionar un amplio conjunto de escenarios de entrenamiento y el tiempo requerido para su diseño. Otro problema destacado por uno de los participantes es relativo a la motivación de los estudiantes: éstos pueden perder interés cuando interaccionan con Agentes Virtuales. "*Si esto ocurre, se van a esforzar menos. Es importante que los estudiantes sean conscientes en todo momento de esto para que puedan sacar el máximo provecho del uso del entorno*". Además, el Agente Guía podría ser demasiado molesto en ciertas ocasiones y entorpecer el flujo de la conversación. Esto significa que la persona encargada del diseño de los escenarios debe buscar un balance entre realismo y objetivos formativos, y debe proporcionar al estudiante la ayuda suficiente para completar el escenario dentro del tiempo límite. Esta ayuda del Agente Guía debe planificarse de forma cuidadosa para evitar molestias innecesarias. Igualmente, antes del inicio de la simulación, se debería trabajar en proporcionar al estudiante una cierta introducción al contexto del problema tratado en el escenario de entrenamiento.

Finalmente, los análisis de los resultados indicaron que VENTURE tenía el potencial de incrementar la confianza para llevar a cabo comunicaciones más efectivas en DGS, y que era capaz de reproducir escenarios realistas, y corregir y guiar al estudiante. Tras el análisis de los resultados, desarrollamos el prototipo de VENTURE que evaluó posteriormente un grupo de expertos (VENTURE V1).

5 Evaluación de expertos

Como parte de la fase 4 de la metodología se aplicó una evaluación heurística como la descrita en [16]. Esta evaluación incluyó dieciocho profesionales e investigadores de trece nacionalidades seleccionados por su experiencia en DGS (ocho años de experiencia de media). El grupo de los profesionales estaba formado por jefes de proyecto, investigadores, jefes de marketing y jefes de calidad y procesos de empresas cuyas sedes representaban cuatro países diferentes.

Los participantes recibieron una explicación verbal de los objetivos de la evaluación y el funcionamiento del simulador de chat y el diseñador de escenarios. Tras ello, ejecutaron un breve escenario de entrenamiento usando el simulador de chat.

Durante la simulación, los participantes jugaron el rol de desarrolladores software interaccionando con un desarrollador virtual de nacionalidad española. Este desarrollador cometía errores típicos de los españoles interaccionando en inglés. El Agente Guía explicó dichos problemas a los participantes, que también recibieron ayuda por parte del Agente Guía cuando interaccionaban de manera incorrecta. Por ejemplo, cuando eran demasiado directos a la hora de abordar un asunto conflictivo.

Tras la finalización de la simulación, los participantes pudieron acceder al diseñador de escenarios. De esta forma, pudieron ver cómo los escenarios son definidos como un flujo de actividades secuenciales compuestas de reglas que pueden contemplar diferentes entradas del estudiante. Los participantes examinaron la definición del escenario que ellos mismos habían ejecutado, examinando su configuración y cómo estaban definidas las reglas que ellos habían disparado durante la simulación. Finalmente, los participantes completaron un cuestionario en el que proporcionaron su visión y sugerencias de ambos componentes de VENTURE.

Esta evaluación sirvió para explorar el potencial del entorno ayudando a estudiantes y profesionales a desarrollar destrezas comunicativas requeridas en DGS, y entender mejor cómo aplicar VENTURE en entornos educativos. Este método fue efectivo para detectar mejoras relativas a aspectos técnicos y también para determinar cómo generar contenido que pueda resultar más efectivo.

Los resultados indican que el prototipo causó una impresión positiva en los participantes. La diversidad de los participantes, incluyendo investigadores, profesionales y profesores dio lugar a una amplia variedad de opiniones sobre diferentes aspectos de la propuesta, incluyendo interfaz de usuario, usabilidad, efectividad para entrenar aspectos lingüísticos y culturales y su aplicabilidad en universidades y empresas.

El análisis de los resultados sugirió que los principales objetivos de la herramienta se cumplían y que VENTURE era capaz de reproducir escenarios útiles y significativos con los que proporcionar entrenamiento en DGS. Se detectó que para poder ser aplicable de forma generalizada, debería contar con un amplio abanico que escenarios basados en problemas reales de DGS. Como resultado de este estudio, se implementaron mejoras en el entorno y se desarrolló una base de conocimiento que se puso disponible a la comunidad de DGS para contribuir con sus patrones y problemas relacionados con DGS (VENTURE V2). Este conocimiento sería útil para generar escenarios de entrenamiento basados en problemáticas reales de las empresas.

6 Estudio de campo

Como parte de la fase 5 de la metodología se llevó a cabo un estudio de campo. El estudio involucró treinta y cuatro participantes de diez nacionalidades distintas. Las nacionalidades mejicana e india fueron excluidas puesto que los escenarios pretendían proporcionar entrenamiento para interaccionar con estas culturas. Dieciséis estudiantes en Ingeniería Informática sin conocimiento en DGS y seis profesionales y doce investigadores con experiencia en DGS participaron en el estudio.

Los participantes recibieron una explicación escrita de los objetivos de la evaluación y los escenarios de entrenamiento. Se diseñaron dos escenarios de entrenamiento, y se dividió a los participantes en dos grupos: G1 y G2. Ambos grupos ejecutaron los mismos escenarios pero en el orden inverso.

Inicialmente, los grupos rellenaron un *Cuestionario Base* como primera tarea del estudio. Tras ello, G1 rellenó el *Pre Cuestionario de Entrenamiento del Escenario A*, y entonces ejecutaron el *Escenario A*, y finalmente rellenaron el *Post Cuestionario de Entrenamiento del Escenario A*. Tras finalizar las tareas para el *Escenario A*, el mismo proceso se siguió con el Escenario B. Por otra parte, la primera tarea de G2 consistió en rellenar el *Pre Cuestionario de Entrenamiento del Escenario B*, ejecutar el *Escenario B* y finalizar con el *Post Cuestionario de Entrenamiento del Escenario B*, y a continuación hacer lo propio con el Escenario A.

El orden de las sesiones se cambió entre los diferentes grupos con el objetivo de comprobar si había alguna diferencia en la dificultad de los escenarios que habría que considerar en el análisis de resultados.

El objetivo del estudio de campo es comprobar si los participantes realmente aprenden de manera eficaz cuando interaccionan con los Agentes Virtuales. En concreto se quiere determinar si usando VENTURE es posible mejorar el conocimiento lingüístico, cultural y el relativo a problemas de DGS. Para ello se analizó la información recogida a través de cuestionarios y también los datos de las evaluaciones automáticas. Los resultados permitieron concluir que los tres puntos mencionados: conocimiento lingüístico, cultural y el relativo a problemas de DGS, se pueden entrenar usando VENTURE.

El análisis también mostró que tras concluir el curso, los participantes habían percibido una mejora en su conocimiento sobre problemas de DGS. También reconocieron haber adquirido nuevas destrezas para afrontar ciertos problemas del DGS.

Los cuestionarios también sirvieron para verificar que el simulador de chat es una buena solución para proporcionar entrenamiento en DGS, y para completar la evaluación se midieron factores relativos a la calidad y usabilidad. Se percibió que el sistema era usable, fácil de usar y claro. Las percepciones del portal de *e-learning*, el simulador de chat y el método de evaluación fueron positivas, lo que significa un importante aspecto si pretendemos que tanto estudiantes como instructores acepten el entorno. De igual forma, de acuerdo al criterio de los participantes, VENTURE tiene el potencial para poder ser aplicado tanto en industria como en academia, de acuerdo también a anteriores resultados.

Tras analizar estos resultados, se descubrieron áreas de trabajo futuro para mejorar la "robustez" y aplicabilidad de VENTURE en entornos educativos. Una de las sugerencias en este respecto fue adaptar el concepto del simulador para poder

integrarlo con otras herramientas utilizadas por las empresas, lo que podría ser un paso importante hacia la aplicación de VENTURE en la industria.

Finalmente, gracias a este estudio, se reconocieron otros puntos débiles e ideas de mejora que serán también parte del trabajo de mejora futuro.

7 Estudio comercial

El objetivo de este último estudio de la fase 5 de la metodología consistió en comprobar si VENTURE es competitivo y proporciona una solución única en el mercado. Además, se pretendía obtener la opinión de expertos en el mercado para orientar el futuro trabajo en lo relativo a la comercialización. Partiendo de la Versión 2 de VENTURE, los objetivos concretos del estudio comercial fueron: 1) analizar el mercado, 2) identificar potenciales clientes, 3) caracterizar el mercado objetivo, 4) definir el modelo de negocio y 5) identificar nuevos requisitos. Los principales resultados de este estudio se agrupan en las siguientes preguntas de investigación:

P1. ¿Qué necesidades formativas tienen las empresas multinacionales en áreas de comunicación e interacción?

Este estudio ha encontrado que para las empresas multinacionales las diferencias culturales son más relevantes que las lingüísticas. Otras áreas de entrenamiento relacionadas con DGS e interesantes para las empresas incluyen: diversidad, asertividad, liderazgo y negociación. Adicionalmente se detectaron otros mercados, no directamente relacionados con DGS, en los que es necesario interaccionar con otras culturas y donde es importante proporcionar entrenamiento específico sobre cómo interaccionar de manera apropiada cumpliendo reglas organizacionales y culturales.

P2. ¿Qué métodos de formación están empleando las empresas actualmente?

La diversidad cultural es parte común de los programas de entrenamiento en las empresas. Estos programas se llevan a cabo a través de clases tradicionales, portales de *e-learning*, actividades de *role-playing*, entornos virtuales 3D y a través de videos que muestran cómo interaccionar y comunicarse en entornos globalizados.

P3. ¿Tiene VENTURE el potencial para cubrir alguna de las necesidades del mercado?

VENTURE tiene el potencial de ser un buen sustituto de las tradicionales actividades de *role-playing*. Los escenarios actuales de *role-playing* empleados por las empresas, podrían traducirse en forma de escenarios de entrenamiento de VENTURE, evitando la necesidad de instructores y automatizando la evaluación, lo que significaría un ahorro considerable de tiempo y costes. Adicionalmente, VENTURE podría competir en coste con los entornos 3D actuales, los cuales requieren una inversión considerable para adaptar el entrenamiento a necesidades específicas.

Como parte de este estudio, se analizaron puntos problemáticos que las potenciales empresas cliente sufren y que están cubiertos por VENTURE. Todos los participantes confirmaron que proporcionar evaluaciones precisas e inmediatas a nivel individual es un punto clave. La habilidad para proporcionar un informe de

resultados automático se consideró como muy importante. Igualmente, la habilidad para generar escenarios de entrenamiento de forma fácil a través del diseñador de escenarios se consideró un factor crítico para la comercialización.

P4. ¿Cómo podría VENTURE ser una opción más atractiva desde el punto de vista de la comercialización?

Colaborar con otras empresas que puedan proporcionar contenido y tecnología sería un punto importante para lograr una mayor ventaja competitiva, tal y como sugiere uno de los entrevistados. El objetivo de dicha colaboración podría ser: 1) integrar la tecnología en herramientas de comunicación existentes, y 2) generar escenarios que consideren la experiencia contrastada en problemas culturales y lingüísticos. Concretamente, sería de especial relevancia considerar actividades de *role-playing* que las empresas puedan estar aplicando en la actualidad.

Otro punto clave que surgió de las entrevistas consistió en que sería preciso limitar el mercado objetivo identificando las culturas más críticas dentro de los sectores prioritarios.

Finalmente, los siguientes puntos resumen los factores clave más importantes para cumplir los requisitos descubiertos en este estudio:

- El coste de introducir la nueva plataforma debe ser reducido, así como el esfuerzo para la generación de nuevo contenido y costes de mantenimiento.
- Capacidad para adaptar la solución a necesidades concretas de los clientes. Por ejemplo, determinadas empresas pueden necesitar entrenar problemas que tengan documentados.
- Cumplir normativas: las empresas suelen estar sujetas a normativas por las cuales necesitan demostrar que proporcionaron entrenamiento en un determinado área y que la persona que recibió el entrenamiento asimiló el contenido. Por tanto, necesitan proporcionar entrenamiento de forma que se pueda registrar la trazabilidad y probar que hubo aprendizaje.
- De acuerdo a un estudio previo [17], y tal y como se descubrió en este trabajo, las empresas generalmente no quieren adaptar nuevos entornos o sistemas. En el área de la formación quieren soluciones integradas, patrones, soluciones y materiales orientados a entrenar problemas concretos.

8 Conclusiones y Trabajo Futuro

Este artículo presenta la metodología de investigación que se ha aplicado para diseñar, desarrollar y evaluar VENTURE. Dadas las características de esta investigación, se ha seguido un método de investigación dividido en varias fases, puesto que se considera un método efectivo para desarrollar un entorno basado en investigación.

Dos Revisiones Sistemáticas de la Literatura sirvieron para enfocar el tema de investigación y para descubrir requisitos e ideas para el diseño inicial del entorno. Una evaluación de viabilidad inicial sirvió para validar el concepto, mientras que una evaluación de expertos posterior al desarrollo del prototipo permitió obtener sugerencias tempranas de participantes que usaron el entorno. El último paso de la metodología consistió en evaluar si es posible aprender DGS mediante el uso de VENTURE a través de un curso completo de DGS; y también se evaluaron las opciones comerciales del entorno.

Descubrir lo que el mercado de la formación en DGS está demandando y ofreciendo es clave para planificar los pasos futuros de la investigación y desarrollo del entorno. Con la aplicación de esta metodología, nos reunimos con empresas que nos proporcionaron ideas sobre el potencial de VENTURE y sus opciones comerciales. Las recomendaciones que se derivan de este estudio sugieren que cuando se trabaja en proyectos de investigación factores como el costo, personalización e integración con herramientas existentes deben tenerse en cuenta. Para trasladarse de un marco teórico hacia una herramienta que pueda tener salida comercial es preciso que la idea sobre la que se sustenta quede claramente diferenciada no solo de la investigación existente, sino también del resto de ofertas comerciales.

Aplicar esta metodología basada en fases nos facilitó una mejora iterativa de VENTURE para adaptarlo tanto a los requisitos de potenciales usuarios como a las necesidades del mercado. Por tanto, recomendamos la aplicación de esta metodología para investigadores que pretendan orientar sus modelos y herramientas hacia un entorno práctico.

A través de la metodología se reconocieron problemas reales y puntos problemáticos para las empresas, y se descubrieron potenciales requisitos y retos gracias a la colaboración con la industria y la academia. Estos resultados fueron de ayuda para definir el trabajo futuro.

Los esfuerzos futuros se centrarán en proporcionar una versión comercial de VENTURE, mejorando el factor coste-eficacia del simulador de chat, el simulador de e-mails, el diseñador y la plataforma *e-learning* implementando las ideas y mejoras analizadas en este artículo. Generar un número representativo de escenarios de entrenamiento será una de las áreas más importantes en las que se centrará el trabajo. Estos escenarios deben reproducir problemas concretos que ocurren en empresas.

Los principales objetivos de mejora de la eficiencia y robustez requeridos por el mercado, deberán centrarse en evitar potenciales situaciones en las que el estudiante se pueda quedar bloqueado intentando expresar algo en una manera que el Agente Virtual no está esperando. Para evitar este problema, se debe trabajar en mejorar la inteligencia de los Agentes Virtuales de manera que sean capaces de detectar un mayor abanico de posibilidades de interacción de los estudiantes de manera autónoma.

Agradecimientos. Este trabajo está financiado en parte por Science Foundation Ireland (10/CE/I1855, Lero - the Irish Software Engineering Research Centre (www.lero.ie)). También está financiado por el proyecto GEODAS-BC (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01), por el proyecto GLOBALIA (PEII11-0291-5274), Consejería de Educación y Ciencia, Junta de Comunidades de Castilla-La Mancha, y por el proyecto ITEA-2 (12012) - SDGear - Software Development Governance as a Service.

Referencias

1. Monasor, M.J., Piattini, M., Vizcaíno, A.: Challenges and Improvements in Distributed Software Development: A Systematic Review. Advances in Software Engineering 1-16 (2009)
2. Damian, D., Marczak, S., Dascalu, M., Heiss, M., Liche, A.: Using Real-Time Conferencing Tools to Leverage the Collaborative Potential of Global Teams: An Experience Report from Siemens IT Solutions and Services. Fourth IEEE International Conference on Global Software Engineering (ICGSE), pp. 239-243. IEEE Computer Society, Limerick, Ireland (2009)
3. Monasor, M.J., Vizcaíno, A., Piattini, M., Caballero, I.: Preparing students and engineers for Global Software Development: A Systematic Review. International Conference on Global Software Development (ICGSE), pp. 177-186. IEEE Computer Society, Princeton, NJ, USA (2010)
4. Petkovic, D., Thompson, G.D., Todtenhoefer, R.: Assessment and comparison of local and global SW engineering practices in a classroom setting. Proceedings of the 13th annual conference on Innovation and technology in computer science education, pp. 78-82. ACM, Madrid, Spain (2008)
5. Deitersy, C., Herrmannz, C., Hildebrandtz, R., Knauss, E., Kuhrmannx, M., Rauschy, A., Rumpez, B., Schneider, K.: GloSE-Lab: Teaching Global Software Engineering. International Conference on Global Software Engineering (ICGSE), pp. 156-160, Helsinki, Findland (2011)
6. Lutz, B.: Training for Global Software Development in an International "Learning Network". Proceedings of the International Conference on Global Software Engineering, pp. 140-150. IEEE Computer Society (2007)
7. Charles, M., Marschan-Piekkari, R.: Language Training for Enhanced Horizontal Communication: A Challenge for MNCs. Business Communication Quarterly 65, 9-29 (2002)
8. Monasor, M.J., Vizcaíno, A., Piattini, M.: Cultural and linguistic problems in GSD: a simulator to train engineers in these issues. Journal of Software Maintenance and Evolution: Research and Practice (Special Issue on Global Software Engineering) 24, 707-717 (2011)
9. Monasor, M.J., Vizcaino, A., Piattini, M.: VENTURE: Hacia un entorno para el entrenamiento del Desarrollo Global de Software. XVI Jornadas de Ingeniería del Software y Bases de Datos (JISBD), pp. 1065-1070, A Coruña (Spain) (2011)
10. Monasor, M.J.: A framework for interaction training in Global Software Development. pp. 219. University of Castilla-La Mancha, Ciudad Real (2014)
11. Monasor, M.J., Vizcaíno, A., Piattini, M., Noll, J., Beecham, S.: Simulating Global Software Development processes for use in Education: A Feasibility Study. In: McCaffery, F., O´Connor, R.V., Messnarz, R. (eds.) 20th European Conference, EuroSPI, vol. 364, pp. 36-47. Springer, Dundalk, Ireland (2013)
12. Monasor, M.J., Noll, J., Vizcaíno, A., Piattini, M., Beecham, S.: Walk before you run: using heuristic evaluation to assess a training tool prototype. 18th International Conference on Evaluation and Assessment in Software Engineering (EASE), pp. to appear, London, UK (2014)

13. Monasor, M.J., Vizcaíno, A., Piattini, M., Noll, J., Beecham, S.: Assessment process for a simulation-based training environment in Global Software Development. 19th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE), pp. to appear, Uppsala, Sweden (2014)
14. Johnson, R.B., Onwuegbuzie, A.J., Turner, L.A.: Toward a Definition of Mixed Methods Research. Journal of Mixed Methods Research 1, 112-133 (2007)
15. Clark, V.L.P.: The Adoption and Practice of Mixed Methods: U.S. Trends in Federally Funded Health-Related Research. Qualitative Inquiry 16, 428-440 (2010)
16. Beecham, S., Carroll, N., Noll, J.: A Decision Support System for Global Team Management: Expert Evaluation. REMIDI - International Workshop on Tool Support Development and Management in Distributed Software Projects co-located with 7th IEEE International Conference on Global Software Engineering (ICGSE), pp. 12 - 17. IEEE Computer Society, Porto Alegre, Brazil (2012)
17. Beecham, S., Oleary, P., Richardson, I., Baker, S., Noll, J.: Who Are We Doing Global Software Engineering Research For? In: International Conference on Global Software Engineering (ICGSE), pp. 41-50.

Sim-XPerience: Simulación Basada en Agentes Aplicada al Desarrollo de Software con XP

Cristina Capitas[1], Nuria Hurtado[2], Mercedes Ruiz[2]

Grupo de Investigación SPI&FM
Departamento de Ingeniería Informática
Universidad de Cádiz
C/ Chile nº1- 11003 – Cádiz
(Teléfono: +34 956 015296) (fax: +34 956 015139)
[1]cristina.capitassanchez@mail.uca.es
[2]{nuria.hurtado, mercedes.ruiz}@uca.es

Resumen. En los últimos años han ganado protagonismo los métodos de desarrollo ágil, especialmente indicados en proyectos con requisitos poco definidos o cambiantes. Dentro de las metodologías ágiles eXtreme Programmimg (XP) es una de las más conocidas y mejor definidas. Sin embargo, uno de los factores que complica su aplicación es la dificultad de realizar estimaciones en este tipo de proyectos. En este trabajo, se describe el desarrollo de Sim-XPerience: un modelo de simulación aplicado al desarrollo de software bajo la metodología XP, que, a diferencia de los que podemos encontrar en la literatura, se ha desarrollado siguiendo el paradigma basado en agentes. Sim-XPerience se plantea como una herramienta de ayuda a la toma de decisiones para gestores y/o directores de proyectos que se desarrollen bajo esta metodología ágil. A través de los parámetros de entrada del modelo es posible configurar las características específicas del proyecto que se desee simular, y experimentar diferentes decisiones relativas a la gestión del equipo y la asignación de tareas, observando la evolución del proceso y del equipo así como las desviaciones de tiempo y coste respecto a las estimaciones iniciales. De este modo, Sim-XPerience constituye una valiosa herramienta de experimentación en el ámbito del desarrollo de software bajo la metodología XP.

Palabras clave: Modelado y Simulación, Programación Extrema, Metodologías Ágiles, Proceso Software.

1 Introducción

Las métodologías ágiles de desarrollo de software han cobrado gran relevancia en los últimos años tanto en ámbitos académicos como empresariales. Cada vez son más las organizaciones que adoptan alguno de estos métodos o están interesadas en hacerlo. Estos métodos surgen como reacción de la comunidad a las metodologías más clásicas

que han dominado el mundo de la ingeniería del software [1]. Las metodologías tradicionales se basan en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Aunque este enfoque ha demostrado ser efectivo en proyectos de gran tamaño, no resulta ser el más adecuado para aquellos proyectos con requisitos poco definidos o cambiantes y donde se exige reducir drásticamente los tiempos de desarrollo sin perder de vista la calidad [2]. Las metodologías ágiles estan especialmente orientadas a proyectos pequeños aportando una gran simplificación. En ellas se le dá mayor valor a las personas, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones cortas [3][4].

Una de las metodologías ágiles más conocidas y aplicadas es eXtreme Programming (XP), precursora de las metodologías ágiles [5][6]. XP está impulsado por un conjunto de valores compartidos que incluyen simplicidad, comunicación, retroalimentación y coraje.

Es posible encontrar numerosos libros, revistas y artículos que estudian la efectividad de las prácticas de XP, pero la mayoría de la investigación en esta línea se basa en estudios empíricos, los cuales consumen gran cantidad de recursos. Hay poca evidencia empírica con base científica que apoye la aplicación del método XP y poca investigación que capte las relaciones entrelazadas entre las diferentes prácticas de XP [7][8]. Por otro lado, debido a la naturaleza cambiante de los proyectos y la escasa documentación que se realiza sobre los mismos resulta difícil realizar estimaciones de forma anticipada [9].

En estos entornos cambiantes, en los que XP está especialmente indicado, resultan especialmente útiles las técnicas de modelado y simulación ya que, a través de ellas, será posible proporcionar a los gestores de proyectos un modo de experimentar las diferentes configuraciones y entender los efectos de diferentes políticas o estrategias, sin la necesidad de asumir los riesgos y costes que una experimentación real conlleva. Al mismo tiempo, estas herramientas sirven de ayuda para realizar estimaciones anticipadas de los proyectos. En esta línea, en la literatura podemos encontrar algunos modelos de simulación, que posteriormente comentaremos, que analizan el efecto de algunas de las prácticas de XP en el desarrollo de software. Los modelos desarrollados hasta el momento, utilizan en su mayoría el enfoque de simulación basado en la dinámica de sistemas.

En este artículo, presentamos Sim-XPerience, un modelo de simulación que, a diferencia de los desarrollados en la literatura, se ha implementado bajo el paradigma de simulación basado en agentes. Está orientado a servir como herramienta de ayuda a la toma de decisiones para los gestores y/o directores de proyectos que sigan la metodología XP. Sim-XPerience presta especial atención a los aspectos relativos al equipo de desarrollo y su evolución durante el proyecto, cubriendo uno de los principios de la metodología XP que centra su atención en las personas.

El modelo que se presenta en este trabajo simula una iteración del proceso permitiendo configurar un conjunto de parámetros de entrada que permiten adaptar el modelo a las carácterísticas específicas del proyecto a simular. Los resultados de una iteración podrán servir, si se desea, para alimentar la entrada de la siguiente iteración del modelo y, de ese modo, simular bien una parte o un proyecto completo.

Sim-XPerience permite también configurar el equipo de trabajadores y sus principales parámetros, así cómo la información relativa a las tareas en las que se desglosa una determinada historia de usuario.

Durante la simulación, Sim-XPerience muestra de forma dinámica, a través de una animación, el comportamiento de las diferentes parejas del equipo de trabajadores así como la evolución de la experiencia del equipo y del desempeño de las tareas a lo largo del tiempo. La simulación del modelo permite obtener información acerca de los factores clave del proyecto bajo diferentes estrategias de asignación de tareas.

La estructura del artículo es la siguiente: en el apartado 2 se comentan los trabajos relacionados sobre la aplicación de las técnicas del modelado y simulación al desarrollo de software con XP, el apartado 3 explica la metodología que se ha seguido para desarrollar Sim-XPerience, el apartado 4 detalla el desarrollo del modelo siguiendo las fases de la metodología explicada en el apartado 3, en la fase de simulación se explicará el caso de uso elegido para mostrar los resultados de la simulación del modelo. Finalmente, se presentan en el apartado 5 las conclusiones y el trabajo futuro a desarrollar en este ámbito.

2 Trabajos relacionados

Las técnicas de modelado y simulación se han aplicado desde principios de los 90 para dar respuesta a distintas cuestiones relacionadas con el proceso software, considerándose herramientas valiosas en las tareas de gestión y de toma de decisiones [10]. En este apartado comentaremos los trabajos más relevantes relacionados con la aplicación de estas técnicas en el ámbito del desarrollo de software con XP.

Kuppuswami [11] propone un modelo de simulación dinámico del proceso de desarrollo de XP para mostrar la naturaleza constante de la curva de coste de cambio. El trabajo describe los pasos a seguir para construir la curva usando el modelo. En [12] el mismo autor desarrolla un modelo de simulación dinámico, en este caso con el objetivo de analizar el efecto de las prácticas XP en el esfuerzo de desarrollo.

Vojislav [13] investiga el uso de la Dinámica de Sistemas para modelar, simular y analizar la práctica de la programación por parejas. El modelo permite la exploración de algunas de las variables que afectan a la eficiencia de la programación en parejas. Los autores concluyen que la competencia en XP aumenta con la compatibilidad psicológica entre la pareja y la velocidad del adaptación entre los miembros de la pareja.

Marco Melis [8] desarrolla un modelo de simulación híbrido, de eventos discretos y Dinámica de Sistema, centrado en dos prácticas de XP: Programación por Pares (PP) y Pruebas antes de programar (TDD). Con datos contrastados con proyectos reales, lanza el modelo bajo las posibles combinaciones de estas dos prácticas.

Kasiak y Godoy [14], desarrollaron un modelo dinámico con VenSim PLE, enfocado al subsistema del desarrollo de tareas, que agrupa las variables involucradas en un proyecto llevado a cabo con XP. El modelo permite analizar el efecto de la implementación conjunta de las prácticas de dicha metodología y ayuda en la gestión de este tipo de proyectos.

En [7] Yong describe un modelo de dinámica de sistemas del proceso de desarrollo de software con XP que puede ser usado para evaluar cuantitativamente el proceso software y estudiar los efectos de las prácticas XP a través de la simulación.

Sim-XPerience, al igual que [7] tiene el objetivo de servir como herramienta de ayuda en la toma de decisiones pero centrándose de manera muy especial en el comportamiento del equipo de trabajadores y permitiendo al usuario experimentar bajo diferentes estrategias de asignación de tareas.

Todos los modelos de simulación comentados siguen, al menos en parte, el enfoque de simulación basado en la dinámica de sistemas en el cuál, el proceso de abstracción se realiza sobre el sistema en su conjunto centrándose en las relaciones causales que ligan las variables observables. Sim-Xperience, sin embargo, se ha realizado siguiendo el paradigma basado en agentes centrándose en los componentes básicos del sistema, sus propiedades individuales y la interacción entre ellos dejando que el comportamiento global del sistema emerja como consecuencia de las condiciones impuestas sobre sus componentes [15]. Dado que uno de los objetivos de Sim-Xperience es observar la evolución de las parejas de desarrolladores, el enfoque basado en agentes resulta especialmente adecuado.

3 Metodología de desarrollo

Existen diferentes metodologías para el desarrollo de modelos de simulación [166][17][18][19]. En este trabajo se ha tomado como referencia el proceso consolidado para SPSM (Software Process Simulation Modelling) [19], que consta de las siguientes etapas:

1. Definición del problema
2. Elección del paradigma de simulación
3. Elección de la herramienta de simulación y técnicas
4. Diseño del modelo
   (a) Definir alcance del modelo. Ámbito
   (b) Determinación del propósito
   (c) Identificación de las variables resultado
   (d) Identificación de los parámetros de entrada
   (e) Conceptualización
5. Implementación del modelo
6. Verificación, calibración y validación del modelo
7. Simulación y análisis de resultados
8. Documentación

La definición del problema es el punto inicial del SPSM e influirá en aspectos importantes como el entorno de la simulación o el nivel de detalle a tener en cuenta. Una vez definido el problema, se procede a la elección del paradigma de simulación que, en ocasiones, puede venir determinado por el propósito y ámbito del mismo, ya que dependiendo del tipo de problema y su nivel de abstracción, será más o menos recomendable el uso de un determinado paradigma. La elección de la herramienta

dependerá, en gran medida, del paradigma seleccionado, con excepción de aquellas herramientas que sean multi-paradigma.

El diseño es, posiblemente, la etapa más compleja y requiere que el/los modelador/es conozcan con detalle el funcionamiento del sistema y las diferentes aproximaciones para modelar las partes y procesos del mismo. En primer lugar, se definen ámbito y alcance del modelo acotando el problema a modelar, el propósito estará directamente relacionado con el ámbito del modelo y a su vez las variables resultado, que darán respuesta a las preguntas que debe resolver el modelo, estarán directamente relacionadas con el propósito determinado. Los parámetros de entrada servirán para alimentar el modelo y definir los experimentos.

Durante la fase de conceptualización en un modelo de simulación basado en agentes debemos identificar los componentes básicos del sistema real, y abstraer sus propiedades individuales y las interacciones que se producen entre ellos [15].

En la fase de implementación se traduce el modelo conceptual a un modelo computacional, con el fin de utilizarlo para obtener datos experimentales, pudiendo usar para ello diferentes herramientas.

La verificación, calibración y validación de los modelos de simulación es un aspecto que merece mucha atención. La verificación significa asegurar que el modelo computacional es correcto y está correctamente implementado. La validación de la estructura del modelo y de los resultados de simulación consiste en asegurar que el modelo representa adecuadamente el sistema en estudio y se comporta de forma esperada [20][21]. Finalmente, la calibración del modelo suele darse de manera simultánea a la validación, independientemente de la estrategia de validación utilizada.

En la fase de simulación y análisis de resultados se diseñarán los experimentos a realizar con el modelo dependiendo del objetivo de estudio. Se fijarán los valores para los parámetros de entrada y se lanzarán las simulaciones variando los parámetros de estudio en función del objetivo los experimentos. Los datos obtenidos podrán ser objeto, posteriormente, de análisis estadístico.

Por último, aunque no por ello menos importante, se encuentra la etapa de documentación, en la que se trata de dejar constancia en un documento formal, de todo el proceso de desarrollo del modelo.

4 Desarrollo del modelo

La explicación del desarrollo del modelo seguirá el esquema propuesto anteriormente en el apartado 3.

La dificultad existente para realizar estimaciones en proyectos que sigan la metodología XP, **planteó** la necesidad de poder experimentar diferentes decisiones relativas a la gestión de proyectos que siguen metodología XP. Para ello, se propone crear SimXPerience. Se ha elegido para el desarrollo de la herramienta el **paradigma de simulación** basado en agentes, ya que uno de sus objetivos es cubrir aspectos relacionados con el equipo de trabajadores y su gestión, teniendo en cuenta factores de experiencia y motivación y con el objetivo, entre otros, de poder observar el comportamiento y la evolución de cada pareja del equipo durante las sucesivas iteraciones del proyecto. El

paradigma de simulación basado en agentes resulta adecuado para satisfacer estos requisitos al permitir la construcción de modelos de sistemas complejos y descentralizados con reglas sofisticadas para gobernar el comportamiento e interacciones de cada uno de los agentes. El modelo se ha implementado a través de la **herramienta de simulación** Anylogic™ [22].

El **ámbito** de Sim-XPerience es el de una iteración del proyecto. El modelo se centra en simular iteraciones del proyecto independientes, suponiendo equipos de desarrollo de un máximo de diez personas y un mínimo de dos, puesto que una de las prácticas esenciales de dicha metodología es la programación en parejas. No obstante, el modelo permite almacenar los datos de salida de una iteración con el objetivo de servir de entrada para la siguiente y así poder simular todas las iteraciones de un proyecto completo.

La simulación de cada iteración de Sim-XPerience tiene el **propósito** de mostrar la evolución del equipo de desarrolladores así como algunos de los factores clave de cualquier proyecto de desarrollo de software, como son coste y tiempo. Sirviéndose de los datos de entrada, el modelo muestra la desviación de tiempo y de coste con respecto a lo estimado y, además, la evolución de la curva de aprendizaje del equipo de desarrollo, y la asignación y evolución del desarrollo de las tareas de cada pareja del equipo. Para implementar otra de las prácticas de XP, se ha considerado que los desarrolladores no trabajan más de 40 horas semanales (8 horas/día laborable). También se ha tenido en cuenta que las tareas en las que cada historia de usuario se descompone, se pueden agrupar en cuatro áreas principales: análisis, diseño, desarrollo y pruebas.

Sim-XPerience contempla las siguientes **variables resultado**:

- *Tiempo real de la tarea*: Duración de la tarea medida en días.
- *Tiempo real de la iteración*: Tiempo de duración de la iteración medido en días.
- *Incremento de la Experiencia del trabajador en cada una de las áreas*
- *Incremento de la Experiencia media del equipo*
- *Coste (euros mensuales)*: Coste de la iteración.

Los **parámetros de entrada** del modelo pueden clasificarse en seis categorías: relativos a los trabajadores, a los costes, a la iteración, a las tareas, otros parámetros generales del modelo y por último los relativos al lanzamiento de la simulación. A continuación se detalla cada una de estas categorías:

Parámetros relativos a los trabajadores

- Nombre del trabajador.
- Nivel de experiencia inicial. Indica el nivel inicial de experiencia de cada trabajador, en las distintas áreas del proceso de desarrollo software.

Parámetros relativos a los costes

- Salario de trabajadores (euros/mes). Los trabajadores de desarrollo de software se han catalogado en:
  — *Trabajador Junior*. Trabajador con una experiencia inferior a dos años.

— *Trabajador Semi-Senior*. Trabajador con una experiencia inferior a cuatro años.
— *Trabajador Senior*. Trabajador con una experiencia superior a cuatro años.

Parámetros relativos a la iteración

- *Número de la iteración*. Número de la iteración que se vaya a simular.
- *Duración estimada*. La iteración tendrá una fecha de entrega fijada, por lo que se tendrá una duración total estimada teniendo en cuenta las recomendaciones de XP.
- *Número de niveles de experiencia*. La elección del número de niveles fijará el rango de experiencia, éste podrá estar entre 1 y 3 (3 niveles) o entre 1 y 5 (5 niveles).
- *Probabilidad de retraso por motivos externos (probabilidad entre 0 y 1, ambos inclusive)*. En este parámetro indicará la probabilidad de que surjan retrasos.
- *Periodicidad del retraso por motivos externos (en días)*. Indica la periodicidad con la que se prevé que pueden surgir los retrasos externos. Puede ser cada día, cada semana (7 días), cada 3 días, etc.

Parámetros relativos a las tareas

- *ID*. Es un número entero que identifica a la tarea.
- *Historia de usuario*. En la planificación de una iteración se realiza el tratamiento de las historias de usuario que se hayan acordado con el cliente para su entrega
- *Área*. Una vez descompuesta la historia de usuario en tareas, éstas pertenecen a un área del proceso de desarrollo software de las comentadas anteriormente.
- *Tiempo estimado*. Las personas que se vayan a hacer cargo de la tarea estiman el tiempo en días (o fracciones de días) que podrían tardar en realizarla.
- *Prioridad*. Es de tipo entero y hace referencia a la prioridad de una historia de usuario respecto a las otras, por lo que todas las tareas que pertenezcan a una misma historia de usuario han de tener la misma prioridad.
- *Dificultad*. Es de tipo entero indicando el valor 1 la máxima dificultad.
- *Tarea de la que depende*. Normalmente suele ocurrir que haya tareas que dependan de otras, incluso tareas que no pueden dar comienzo hasta que otra haya finalizado, como puede ser el caso de realizar pruebas generales por ejemplo. Para indicar cuál es la tarea de la que se depende basta con indicar el identificador de la misma.
- *Descripción*. Este parámetro es aclaratorio sobre la finalidad de la tarea.

Parámetros relativos al lanzamiento de la simulación:

- *Decidir si se desea sobrescribir el fichero de entrada de trabajadores*. En este fichero se encuentran los valores iniciales de los parámetros de entrada relativos a los trabajadores. Sim-XPerience da posibilidad al usuario de elegir si desea actualizar el fichero de entrada de los trabajadores tras la simulación con la intención de que estos datos puedan servir como datos de entrada para la siguiente iteración.
- *Decidir si la experiencia de los trabajadores influirá en el tiempo de realización de las tareas*. Sim-XPerience da la posibilidad de decidir si la experiencia de los trabajadores influirá o no en el tiempo de realización de las tareas.
- *Decidir si contemplar la motivación de los trabajadores a la hora de desempeñar su trabajo*. La motivación de los trabajadores será otro factor que influirá en el

tiempo de realización de las tareas y que se ha implementado a partir de eventos aleatorios.

- *Elegir la estrategia o criterio de asignación de tareas*. La asignación de tareas podrá realizarse bajo dos estrategias distintas, el usuario deberá elegir el criterio bajo el cual se regirá la asignación de las tareas durante la simulación:
  - Estrategia A: consiste en tener en cuenta la experiencia de los trabajadores en el área de la tarea a desempeñar como criterio prioritario.
  - Estrategia B: consiste en intentar realizar un reparto equitativo de trabajo de manera que todos los trabajadores tengan una carga de trabajo igual o similar, se tendrá en cuenta como criterio prioritario el número de tareas realizadas, y como segundo criterio la experiencia, eligiendo para una tarea determinada a aquel trabajador que mayor experiencia tenga en el área de la tarea a desarrollar de entre aquellos que hayan trabajado menos tiempo.

4.1 Conceptualización del modelo

Para la conceptualización del modelo se han considerado dos agentes: **Trabajador** y **Tarea**, los cuales pasan por los diferentes estados reflejados en la Tabla 1.

XP se centra, entre otras cosas, en el equipo de trabajo. Por este motivo, el modelo sigue el diagrama de estados del trabajador representado en la Fig.1, puesto que de su disponibilidad depende que se realicen o no las tareas. No obstante, hay que tener en cuenta que la simulación seguirá ejecutándose mientras haya alguna tarea por asignar. Así, para detallar el mecanismo del funcionamiento del modelo se seguirá dicho diagrama.

Para que un Trabajador pase de estar *Libre* a *Pendiente de asignación*, éste ha de ser válido para la Tarea que en el momento tenga que desempeñarse. El orden de realización de tareas se hace en función de la prioridad asociada, considerando la posible dependencia que pueda existir de otras tareas. Para comprobar si un Trabajador es válido para una Tarea, además de su disponibilidad, se contemplan dos factores: el número de tareas que lleva realizadas y la experiencia en el área a la que pertenece la Tarea.

En función de la estrategia que haya elegido el usuario antes de lanzar la simulación se le dará una mayor prioridad a un factor o a otro. Sim- XPerience da la posibilidad de elegir al usuario para que éste pueda comparar los resultados de ambos criterios. En el caso de haber elegido la Estrategia B se dará prioridad al número de tareas que un trabajador lleve realizadas ante su experiencia, y viceversa en el caso de haber elegido la Estrategia A.

Tabla 1. Estados de los agentes

Agente	Estado	Descripción
Trabajador	Libre	El Trabajador está disponible para que le sean asignadas tareas.
	Pendiente de asignación	Un Trabajador pasará a este estado cuando se le asigne una tarea y permanecerá "pendiente" hasta que dicha tarea tenga asignados dos trabajadores responsables (pareja).

	Trabajando	Cuando la tarea esté asignada, el Trabajador pasa a este estado. Dividido a su vez en cuatro estados diferentes, uno por cada área contemplada.
Tarea	Sin asignar	Estado inicial en el que se encuentran todas las tareas.
	Asignada	La tarea pasa a este estado cuando tiene una pareja de trabajadores asignados.
	Finalizada	Una vez la Tarea haya sido asignada, el tiempo real de ésta se inicia. Cuando éste llegue a su fin, la tarea se dará por finalizada.

Una vez que se determine que un Trabajador es válido para realizar la tarea, se procede a la asignación. Si la Tarea ya tiene un Trabajador asignado, ésta pasará a estar *Asignada* y la pareja de trabajadores pasarán a estar *Trabajando*, en caso contrario, se buscará al próximo Trabajador válido para la misma.

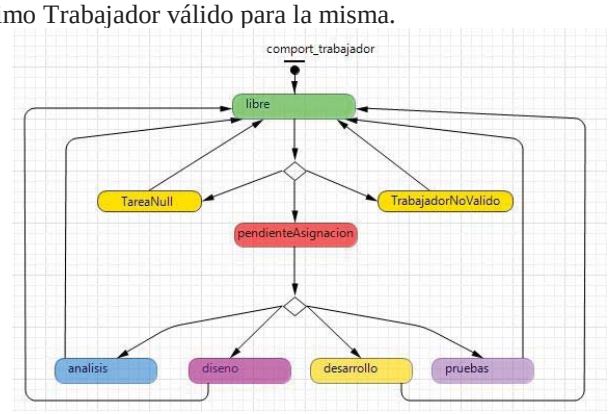


Fig. 1. Diagrama de estados del trabajador

Cuando una Tarea ya tiene a sus dos responsables asignados se procede a considerar la influencia de la experiencia de los Trabajadores. Ésta toma dos vertientes:

1. *Incremento de la experiencia en los trabajadores*. La experiencia de un trabajador se puede ver incrementada desde dos vertientes: por un lado, gracias al hecho de trabajar en pareja, ya que los conocimientos del compañero influye positivamente en la experiencia, y por otro, debido al tiempo dedicado en el área a la que pertenece la tarea que ha desempeñado.
2. *Tiempo de realización de la tarea*. Partiendo de la premisa de que el tiempo de una tarea se estima, considerando valores medios de la experiencia, este se verá alterado en función de cómo sea la experiencia real del trabajador. De manera que si la experiencia de los trabajadores responsables de la tarea está por debajo de la media, el tiempo de la tarea se incrementará, y si está por encima, el tiempo se reducirá. La cuantía de dicha alteración dependerá de la dificultad de la tarea, puesto que a mayor dificultad mayor será la alteración y viceversa.

El tiempo de una Tarea se puede ver alterado a causa de la experiencia de los trabajadores responsables, pero también pueden sufrir alteraciones eventuales, las cuales pueden ser originadas por factores externos y/o por el factor motivación, según lo haya decidido el usuario al lanzar la simulación. Independientemente del origen de la alteración eventual, ésta podrá afectar a tareas que estén en curso, es decir, que se estén realizando, o a tareas que aún estén *sin asignar*. El factor motivación podrá influir también de manera positiva, negativa o neutra sobre el tiempo de las tareas.

Finalmente, un Trabajador pasará de estar *Trabajando* a *Libre* cuando el "tiempo real" de la Tarea haya finalizado. En este momento, además de dar por *Finalizada* la tarea, se tendrá en cuenta el tiempo dedicado en el área, por parte de los trabajadores, para aplicarle el incremento correspondiente en la experiencia.

Una vez que se hayan realizado todas las Tareas, Sim-XPerience mostrará los datos resultantes respecto a experiencia, tiempo y coste. Así, se podrá ver la experiencia resultante de cada uno de los trabajadores en las distintas áreas así como la evolución de la experiencia media del equipo. También se podrán realizar las comparativas entre los tiempos y costes estimados y los obtenidos tras la simulación.

Teniendo en cuenta que Sim-XPerience está pensado para simular cada iteración de un proyecto, éste también muestra la evolución global del equipo, considerando el tiempo de realización y la experiencia media del equipo obtenidas en las iteraciones anteriores.

4.2 Simulación del modelo y análisis de resultados

Para ilustrar la simulación del modelo y sus resultados se plantea el siguiente caso de estudio: estudiar la evolución del equipo de trabajadores así como el tiempo y coste de un proyecto de desarrollo de software con XP cuando se utilizan diferentes estrategias de asignación de tareas. Para ello, se simulará para cada estrategia el desarrollo de diez proyectos de similares características. Se supondrá que cuando un proyecto acaba, el mismo equipo se incorpora al siguiente proyecto, con la diferencia de que su experiencia se habrá visto incrementada. En la simulación de los diez proyectos por cada estrategia, sólo se verá modificada la experiencia del equipo de trabajadores para poder reflejar la evolución de la misma a lo largo del tiempo. Todas las simulaciones han estado sujetas a las mismas premisas iniciales, es decir, los datos: estimación de la duración total, probabilidad de retraso por factores externos y la periodicidad de dichos factores de cada iteración del proyecto, serán similares durante la simulación de los diez proyectos y para las dos estrategias consideradas.

De manera esquemática, los factores contemplados son:

- Influencia de la motivación de manera aleatoria e independiente del criterio de asignación de tareas
- Estimación de tiempos para equipos de trabajadores de experiencia media
- Probabilidad de retraso por motivos externos y periodicidad de los mismos.

Los datos que permanecen constantes para los proyectos son los siguientes:

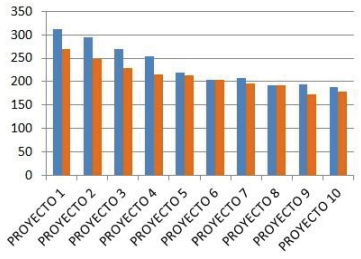
- Un equipo formado por 8 trabajadores.
- 5 niveles de experiencia.
- Salarios de los trabajadores según su categoría. Cada trabajador será catalogado por su experiencia media en todas las áreas:
  - Trabajador Junior: 900 euros al mes.
  - Trabajador Semi-Junior: 1500 euros al mes.
  - Trabajador Senior: 2300 euros al mes.
- Factor motivación. La motivación influirá, positiva o negativamente, de manera aleatoria en los tiempos de ejecución de las tareas, sin considerar la política de asignación de tareas elegida.
- Experiencia de los trabajadores. El tiempo de la tarea se estima teniendo en cuenta una experiencia media. De este modo, los niveles de experiencia que estén por encima o por debajo de la experiencia media, influirán en el tiempo de realización de la tarea, decrementándolo o incrementándolo.

En la tabla 2 se muestran los datos, relativos a cada iteración del proyecto, que se han contemplado al realizar las simulaciones y la estimación de la duración de las tareas de cada una de ellas.

Tabla 2. Datos de cada Iteración del Proyecto

	It 1	**It 2**	**It 3**	**It 4**	**It 5**
Probabilidad de retrasos externos	0.3	0.4	0.5	0.4	0.3
Periodicidad de retrasos externos	1	2	3	5	7
Número de tareas	52	28	24	12	23
Tiempo estimado	70	43	37	48	50

La figura 2 muestra los resultados finales de la simulación para el experimento planteado para las principales variables resultado: tiempo (izquierda) y coste (derecha). Las barras rojas muestran los resultados siguiendo la Estrategia A, que prioriza la experiencia de los trabajadores, y las barras azules corresponden a la Estrategia B, que trata de hacer un reparto más equitativo de las tareas. En esta figura, se puede observar que la Estrategia A proporciona resultados significativamente mejores para los indicadores de tiempo y coste durante los cuatro primeros proyectos que la Estrategia B, momento a partir del cuál los resultados empiezan a estar más equilibrados para las dos estrategias.



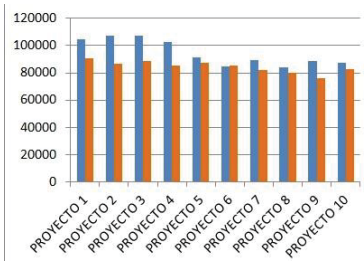


Fig. 2. Resultados para los indicadores de tiempo y coste

Por otro lado la figura 3 representa el porcentaje de mejora en la experiencia media del equipo que presenta la Estrategia B frente a la Estrategia A. Podemos observar que la política de la equidad (Estrategia B) consigue un mayor incremento en la experiencia, llegando incluso en el proyecto 5 a ser un 11% mayor el incremento de la experiencia en la Estrategia B que en la A.

Como conclusión podemos decir que a corto plazo la Estrategia A logra mejores resultados para los principales indicadores del proyecto, pero a partir de cierto momento, para similares resultados de tiempo y coste, obtenemos equipos más formados utilizando la Estrategia B que además está más en consonancia con el espíritu de las prácticas de XP.

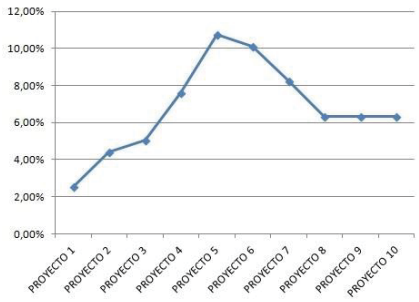


Fig. 3. Porcentaje de mejora en la experiencia del equipo, de la Estrategia B frente a la Estrategia A.

5 Conclusiones y trabajo futuro

Del análisis de la literatura se desprende que los modelos de simulación constituyen valiosas herramientas de ayuda a la toma de decisiones en proyectos de desarrollo de software. En este proyecto se ha desarrollado Sim- XPerience: una herramienta basada en simulación que, bajo el paradigma de simulación basada en agentes, permite la experimentación de diferentes decisiones en la gestión de proyectos de desarrollo de software siguiendo los principios de las metodologías ágiles, concretamente XP. Tal y como sucede en la metodología de referencia, el equipo humano de trabajadores y todo lo relativo a la gestión del mismo constituye una parte importante de esta herramienta. A continuación se destacan a modo de resumen las principales aportaciones de Sim-XPerience:

- Permite al usuario flexibilidad en la configuración de los parámetros de entrada de manera que estos permitan reflejar de las características del proyecto y del equipo de trabajadores.
- Permite la experimentación bajo diferentes estrategias en la asignación de tareas
- Permite variar las estimaciones de tiempo y previsión de retrasos.
- Permite decidir sobre la influencia de la motivación de los trabajadores

- Permite observar la evolución de la experiencia de los trabajadores y del equipo a lo largo del proyecto (curva de aprendizaje)
- Proporciona una interfaz gráfica que muestra a lo largo de la simulación una animación dinámica de la oficina de desarrollo y de los equipos de desarrollo (parejas) trabajando así como la evolución de las diferentes variables resultado.
- Proporciona resultados sobre la evolución de las tareas desarrolladas.
- Proporciona resultados sobre tiempo y coste y desviación de las estimaciones iniciales.
- Proporciona mensajes de advertencia en la interfaz de salida cuando se exceden las estimaciones de coste y tiempo.
- Proporciona comparativas de las variables resultado entre las diferentes iteraciones del proyecto.

Además de las aportaciones particulares de la herramienta, este trabajo aporta como novedad el desarrollo de un modelo, en el ámbito del desarrollo de software con XP, siguiendo un enfoque basado en agentes. Como trabajo futuro a desarrollar en esta línea se pretende afrontar la validación del modelo con datos de proyectos reales. Por otro lado se plantea la incorporación de otros factores clave en cualquier proyecto software como pueden ser los relativos a la calidad.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto amuSE (TIN2013-46928-C3-2-R) del Ministerio de Economía e Innovación y el Fondo Europeo de Desarrollo Regional (FEDER).

Referencias

1. T. Dyba, T. Dingsoyr. What Do We Know about Agile S. D.?. Soft. IEEE, pp. 6-9, 2009.
2. J.H. Canós, P. Letelier, M.C. Penadés. Metodología Ágiles en el Desarrollo Software. VIII Jornadas de Ingeniería de Software y Bases de Datos, JISBD 2003.
3. A. Orjuela, M. Rojas. Las Metodologías Ágiles como una Oportunidad para la Ingeniería del Software Educativo. Revista Avances en Sistemas e Informática.vol5 num 2. 2008.
4. M.A. Awad. A comparison between agile and traditional software development methodologies. M.S. Thesis, University of Western Australia, 2005.
5. K. Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley: Boston, MA. 1999.
6. K. Beck, C. Andres. Extreme Programming Explained: Embrace Change- Second Edition. Addison-Wesley: Boston MA. 2004.
7. Y. Yong, B. Zhou. Evaluating Extreme Programming Effect through System Dynamics Modeling. 978-1-4244-4507-3. 2009.
8. M. Melis, I. Turnu, A. Cau, G. Concas. Evaluating the Impact of Test-First Programming and Pair Programming through Software Process Simulation. Software Process Improvement and Practice. 11: 345–360. 2006.

9. G. Bioul, F. Escobar, M. Alvarez, A. Nardin, E. Ricci Aparicio. Metodologías Ágiles, análisis de su implementación y nuevas propuestas. XVI Congreso Argentino de Ciencias de la Computación CACIC, 2010. pp. 597-606. 2010.
10. M.I. Kellner, R.J. Madachy, D.M. Raffo. Software Process Simulation Modeling: Why? What? How? In: The Journal of Systems and Software, (46: 2), pp. 91-105.1999.
11. S. Kuppuswami, K. Vivekanandan, and P. Rodrigues. A System Dynamics simulation model to find the effects of XP on cost of change curve. In XP2003 Conference Proceedings, pages 54–62, 2003.
12. S. Kuppuswami, K. Vivekanandan, P. Ramaswamy and P. Rodrigues. The effects of individual XP practices on software development effort. SIGSOFT Softw. Eng. Notes, 28(6):6–6, 2003.
13. B. Vojislav, Misic, H. Gevaert and M. Rennie. Extreme dynamics: Modeling the extreme programming software development process. In Proceedings of ProSim04 workshop on Software Process Simulation and Modeling , pages 237–242, 2004.
14. T. Kasiak, D.A. Godoy. Simulación de proyectos de Software desarrollados con XP: Subsistema de desarrollo de tareas. XIV Workshop de Investigadores en Ciencias de la Computación. p. 572-576. 2012.
15. L. R. Izquierdo, J.M. Galán, J.I. Santos, R. del Olmo. Modelado de sistemas complejos mediante simulación basada en agentes y mediante dinámica de sistemas. EMPIRIA. Revista de Metodología de Ciencias Sociales. N.o 16, pp. 85-112, 2008.
16. J. D. Sterman, Business Dynamics. Systems Thinking and Modeling for a Complex World. McGraw-Hill. 2000.
17. I.J. Martinez, G. P. Richardson, Best Practices in System Dynamics Modeling. Plenary paper In Proceedings of the 29th International Conference of the System Dynamics Society, Hines, .H., Diker, V.G. (Eds.), Atlanta, GA, USA, 2001, pp: 1-22.
18. A. M. Law, How to build valid and credible simulation models, Proceedings of the 2009 Winter Simulation Conference, 2009, pp 24-33.
19. N. B. Ali, K.A. Petersen, A consolidated process for software process simulation: State of the art and industry experience, Proceedings of the 38th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2012 , art. no. 6328171 , 2012, pp. 327-336.
20. R. G. Sargent, Verification and validation of simulation models. Proceedings of the 2011Winter Simulation Conference. pp 183-198. 2011.
21. J.A. Sokolowski, C.M. Banks. Principles of Modeling and Simulation. A Multidisciplinary Approach. A John Wiley & Sons, INC., Publication. 2009.
22. Anylogic (Anylogic™) http://www.anylogic.com [último acceso: abril 2014]

Habilidades emocionales en el sector TIC: un análisis a partir de los empleos ofertados en el sector

Víctor Serrano, Juan M. Vara, Esperanza Marcos

Grupo de Investigación Kybele, Universidad Rey Juan Carlos, España
{v.serranob, juanmanuel.vara, esperanza.marcos}@urjc.es

Resumen. Cuando pensamos en los conocimientos que debe tener un profesional de las TIC, pensamos habitualmente en capacidades técnicas: de análisis y diseño de sistemas, captura de requisitos, programación, etc. Sin el factor humano juega un papel clave, ya que la tecnología se construye por y para las personas. Con el fin de comprobar si efectivamente las empresas del sector de las TIC conceden importancia a las capacidades *no técnicas* o habilidades emocionales, este trabajo presenta los resultados de un estudio en el que se analiza la demanda de habilidades emocionales en las ofertas de empleo del sector las TIC, así como la aplicación desarrollada para automatizar la búsqueda, extracción y clasificación de la información analizada en dicho estudio.

Palabras clave: Habilidades emocionales, Análisis de Datos, Liderazgo, Trabajo en equipo, Comunicación.

1 Introducción

Aunque hace mucho tiempo que los trabajos de autores como Thorndike [18], Payne [14] o Gardner [4] reconocen la importancia de los aspectos no cognitivos de la inteligencia, la publicación del libro *Emotional Intelligence* [5] de Goleman supuso una pequeña revolución que popularizó definitivamente la importancia de la *inteligencia emocional*, que podemos definir como una serie de competencias que nos permiten obtener información sobre las emociones y los estados anímicos propios y ajenos y utilizar esa información como guía para determinar nuestro pensamiento y comportamiento [11].

Hasta la publicación del libro de Goleman en 1995, los factores determinantes a la hora de identificar, seleccionar o calificar un trabajador como "trabajador estrella" eran eminentemente los conocimientos y la experiencia del trabajador ligados con la profesión o puesto de trabajo en cuestión. De algún modo, estos factores están directamente relacionados con la formación académica del individuo y vienen determinados por su coeficiente intelectual.

Sin embargo, Goleman [6] defiende que no es el aprendizaje académico el que determina unívocamente el desarrollo profesional del individuo y, así, estudiantes que podrían ser calificados como *en la media*, pueden llegar a destacar profesionalmente por delante de profesionales que han sido brillantes estudiantes. El factor

determinante en estos casos pasa a ser las habilidades propias de la inteligencia emocional, que devienen en lo verdaderamente importante para obtener el máximo rendimiento de cada trabajador en el desempeño de las labores propias de su profesión.

De hecho, la necesidad de desarrollar este tipo de habilidades se destacaba ya en el informe "*Future skills for tomorrow's world*" [1], en el que las principales compañías europeas del sector TIC identificaban los perfiles profesionales que sus organizaciones requerirían en los años subsiguientes y donde destacaban los perfiles creativos, artísticos, entusiastas de la tecnología, matemáticos y científicos, con habilidades de comunicación y capacidad de trabajo en equipo. Dicho informe señalaba igualmente como habilidades sociales más requeridas la capacidad analítica, la atención a los detalles, la iniciativa, la comunicación, la creatividad, la orientación al cliente, el liderazgo, la negociación, la persuasión, la capacidad para relacionarse con otros, la estrategia y la planificación y finalmente el trabajo en equipo.

Son varios los trabajos que más recientemente inciden en algunas de estas observaciones y subrayan una vez más la importancia que van adquiriendo las habilidades emocionales en la selección de aspirantes a un determinado puesto de trabajo, con especial énfasis en el sector TIC [8][10][16].

Todo ello evidencia la necesidad de formar a nuestros graduados en competencias emocionales, frente a la habitual formación centrada en competencias técnicas que tradicionalmente adopta el sector TIC, que produce Ingenieros Informáticos o Ingenieros Software que, por poner algunos ejemplos, saben cómo programar o conocen técnicas de licitación de requisitos, pero adolecen de graves problemas a la hora de hablar en público o de trabajar en equipo [14].

No en vano, los principios del marco educativo que define la implantación del Espacio Europeo de Educación Superior (EEES) [3], ponen el énfasis en una formación completamente orientada a competencias en lugar de la tradicional transmisión de conocimientos [19]. En realidad, la formación orientada a este tipo de competencias no es algo completamente nuevo, hace ya algún tiempo que se ha empezado a introducir la enseñanza de la inteligencia emocional en las escuelas de educación infantil y primaria [16], mientras que ya hace años que se viene aplicando con éxito en las Escuelas de Negocio [19].

Bajo estas premisas y con la intención de comprobar en qué medida este tipo de competencias no técnicas son efectivamente relevantes para los empleadores del sector TIC y reflejar las necesidades de formación de los estudiantes que se convertirán en futuros candidatos para las empresas en los próximos años, el presente trabajo presenta un estudio en el que las ofertas de empleo del sector han sido analizadas para identificar las habilidades emocionales requeridas en las mismas. Para desarrollar dicho estudio (y futuros análisis similares) se ha desarrollado una herramienta (`VSJobStatsAnalyzer`) que permite extraer y analizar automáticamente la información de las diferentes ofertas publicadas.

El resto del artículo se estructura del siguiente modo: la sección 2 describe los objetivos, el ámbito y el diseño del estudio, la sección 3 analiza los resultados obtenidos; la sección 4 introduce la herramienta desarrollada para automatizar la

búsqueda, extracción y clasificación de información y finalmente, la sección 6 resume las principales conclusiones y plantea líneas para el trabajo futuro.

2 Diseño del estudio

En este apartado se definen el ámbito y los parámetros del estudio realizado. Para ello se describen los objetivos y diferentes aspectos del análisis y se discuten sus posibles limitaciones.

2.1 Objetivo y ámbito

El presente estudio se basa en los datos extraídos de las ofertas de empleo del sector TIC publicadas durante una semana laboral (5 días) en el portal de empleo *Infojobs* para las ciudades de Madrid y Barcelona. La selección de *Infojobs* como fuente de información se debe eminentemente al posicionamiento de este portal respecto a otros portales de empleo (según sus responsables, fueron 389.188 las personas que en 2012 consiguieron un empleo utilizando *Infojobs* y acapara más del 60% de la cuota de mercado).

Se analizarán las ofertas publicadas durante los 5 días de una semana para detectar posibles fluctuaciones entre los datos obtenidos al principio o el final de la misma y finalmente aunque tal y cómo se mostrará más adelante (ver Sección 4) la herramienta es capaz de identificar cualquier competencia registrada previamente en un diccionario, inicialmente la búsqueda considerará 21 competencias para luego centrarse en 3 de ellas: liderazgo, trabajo en equipo y comunicación, consideradas las más relevantes en varias de las publicaciones citadas previamente [1][6].

De esta forma, se pretende obtener una visión general lo más objetiva posible sobre la importancia de las habilidades emocionales en general y el liderazgo, el trabajo en equipo y la comunicación en particular, para los ofertantes de empleo en el sector TIC.

Con la intención de ilustrar el alcance del estudio, la Tabla 1 resume el número de ofertas recuperadas por día y ciudad. Estos números dependen obviamente de los criterios de búsqueda seleccionados. Para permitir la selección de valores para estos criterios la herramienta proporciona una serie de controles que serán presentados más adelante junto al resto de la herramienta.

Tabla 1. # Ofertas recuperadas por día y ciudad.

Día	Madrid	Barcelona
Lunes	168	93
Martes	176	82
Miércoles	162	71
Jueves	169	92
Viernes	179	76
Total	**854**	**414**

Finalmente, conviene señalar que para alcanzar el objetivo de este estudio se analizarán diferentes aspectos de la relación entre las capacidades seleccionadas y las características de cada oferta. En particular, de entre las múltiples posibilidades que soporta la herramienta, para el presente análisis se ha optado por estudiar los siguientes aspectos:

- Demanda de las habilidades emocionales objeto de estudio.
- Relación entre el puesto ofertado y las habilidades emocionales requeridas en la oferta.
- Relación entre el rango salarial de la oferta y las habilidades emocionales requeridas en la oferta.
- Relación entre la experiencia y las habilidades emocionales requeridas en la oferta.

2.2 Posibles limitaciones

Esta sección discute brevemente algunas limitaciones del presente estudio, dado que los resultados del mismo podrían diferir en cierta medida si el análisis se realizase en ausencia de estas limitaciones.

En primer lugar, el periodo en el que se realiza el análisis es un factor a tener en cuenta, ya que las grandes corporaciones suelen incorporar personal en determinados momentos del año y por tanto suelen publicar sus ofertas en los meses de Septiembre/Octubre y Febrero/Marzo. Por lo tanto, si la búsqueda de ofertas se realiza fuera de estos periodos se corre el riesgo de no estar considerando las capacidades que estas organizaciones requieren de sus futuros empleados.

Por contra, la contratación por parte de las empresas más pequeñas, o con dificultades económicas no suele responder a patrones pre-establecidos y se limitan a publicar ofertas cuando abordan nuevos proyectos, de tal forma que en un determinado mes pueden publicarse muy pocas ofertas de trabajo, véase Agosto, y al mes siguiente publicarse muchas más. Por todo ello, lo ideal sería realizar un análisis anual o cuatrimestral. En el caso de este trabajo y dada la imposibilidad de realizar todavía un análisis a medio/largo plazo, se ha optado por la semana del 2 al 6 de Septiembre, bajo la creencia de que el regreso de las vacaciones de verano es el periodo escogido por la mayoría de organizaciones para reclutar nuevo personal.

En segundo lugar, el hecho de que las ofertas analizadas sean sólo las ofertas publicadas en el portal *Infojobs* supone otra limitación en sí misma, puesto que, a pesar de ser el más utilizado y el que pública un mayor número de ofertas, no recoge todas las ofertas del sector. No sólo existen otros portales de empleo, sino que en muchos casos son las empresas las que prefieren evitar intermediarios y optan por mantener secciones ad-hoc en sus propios sitios Web o incluso contactan directamente con potenciales candidatos cuando surgen vacantes en su organización. Es evidente que en estos casos el estudio presenta un sesgo en cuanto a las consideraciones relacionadas con las capacidades requeridas por estas empresas.

Por último, existen otras limitaciones de carácter técnico relacionadas con la automatización del análisis que, aunque permite reducir drásticamente el tiempo

dedicado a la extracción de información, puede no resultar tan precisa como una búsqueda manual, ya que obviamente es muy difícil igualar la capacidad de comprensión y análisis del ser humano.

Más concretamente la extracción de datos de las ofertas que implementa la aplicación se basa en un simple *matching*, de manera que se comprueba si en la oferta aparecen determinados términos para determinar si se requiere de una capacidad o no. Esto implica que el proceso de extracción pueda omitir algunos positivos, dado que es posible que una determinada expresión no reconocida por la aplicación haga referencia a una habilidad en concreto. Para paliar este problema se permite la modificación de los diccionarios utilizados por la aplicación, pero hay otros problemas, como la posibilidad de que las ofertas presenten faltas de ortografía que resulten en falos negativos del proceso de *matching*, que son de más difícil solución.

3 Análisis e interpretación de resultados

En esta sección se presentan, analizan e interpretan los resultados obtenidos en el presente estudio, de acuerdo a los objetivos y aspectos a analizar que se fijaron cuando se presentaba el diseño del estudio (Sección 2.1).

3.1 Demanda de habilidades emocionales

En primer lugar, parece conveniente resaltar que el número de ofertas recuperadas y analizadas por la aplicación asciende a 1.268 (ver Tabla 1), lo que a nuestro juicio sirve para evidenciar que, a pesar de las limitaciones comentadas en la sección anterior, se cuenta con una muestra lo suficientemente representativa como para sostener las afirmaciones que se realizarán sobre la información extractada de dichas ofertas.

Sólo el 18.23% de estas ofertas no mencionan explícitamente alguna de las 21 habilidades emocionales consideradas inicialmente en este análisis (ver Anexo). Conviene resaltar que el hecho de que no se mencione explícitamente ninguna habilidad emocional, no implica necesariamente que no se contemplen en el proceso de selección. Es muy probable que la empresa haya optado por que el filtro inicial que supone la recepción de solicitudes en respuesta a la publicación de la oferta se limite sólo a requerir capacidades técnicas para participar en el proceso y analice en una segunda fase de entrevistas las capacidades emocionales de los candidatos.

En lo sucesivo se restringe el ámbito del estudio a 3 habilidades emocionales consideradas particularmente representativas (comunicación, trabajo en equipo y liderazgo) para mostrar un ejemplo del tipo de análisis que soporta la herramienta desarrollada.

3.2 Demanda de las habilidades emocionales objeto de estudio

En esta sección se analiza el volumen de ofertas que demandan cada una las 3 habilidades emocionales que son objeto de estudio.

La más demandada es la capacidad de comunicación, que se menciona en 301 ofertas, triplicando el número de ofertas que demandan habilidades relacionadas con la capacidad de trabajar en equipo (102 ofertas), mientras que la capacidad de liderazgo apenas tiene presencia (14 ofertas).

No obstante, debe tenerse en cuenta que la capacidad de liderazgo es más habitual en puestos de responsabilidad, por lo que un análisis adecuado de la demanda de esta habilidad en las ofertas de empleo del sector pasa por limitar el ámbito del análisis a ofertas para cubrir puestos de manager/gerente o jefe de equipo, de las cuales apenas se incluían entre las ofertas analizadas. Este hecho puede deberse probablemente a que las empresas recurren a procesos de selección internos para la selección de las personas que deben ocupar puestos de responsabilidad, de tal forma que la persona seleccionada habrá demostrado previamente su valía y habilidades, bien porque ya perteneciese a la empresa, o bien por haber interactuado con algunos de sus empleados en el marco de algún proyecto previo.

Respecto a las habilidades de comunicación y trabajo en equipo, una muestra de su relevancia para los empleadores es el hecho de que se demanden en el 34% de las ofertas analizadas. Cabe destacar igualmente que la demanda de estas habilidades no responde a ninguna categorización: aparecen por igual en ofertas con puestos de menor y mayor responsabilidad. De modo que la diferenciación en la demanda de estas capacidades se corresponde con las necesidades puntuales de la empresa a la hora de cubrir un determinado puesto y no se basa en la adecuación de estas habilidades para categorías más exigentes.

3.3 Relación entre el puesto ofertado y las habilidades emocionales requeridas

En esta sección se analiza la relación existente entre las capacidades objeto de estudio y los puestos ofertados.

De este modo, al agrupar las ofertas en las que se demanda al menos una de las 3 habilidades emocionales en función del puesto ofertado, se puede observar que la capacidad de comunicación es especialmente demandada a la hora de cubrir puestos de programador, técnico, consultor y analista, tal como ilustra la Fig. 1(a).

Fig. 1. Relación entre el puesto ofertado y la demanda de las habilidades emocionales objeto de estudio.

En general, estos puestos implican la necesidad de ser capaz de comunicarse eficaz y eficientemente, de cara a evitar el impacto negativo de posibles perdidas o malinterpretaciones de información. Por ejemplo, piénsese en las implicaciones

derivadas de asignar una captura de requisitos a un analista que sea capaz de transmitir de forma apropiada la información recabada a diseñadores o programadores.

Tal como muestra la Fig. 1(b), la capacidad de trabajar en equipo sigue una distribución similar en cuanto a los puestos en los que se requiere, aunque como ya se ha mencionado anteriormente, el número total de ofertas en que se requiere esta habilidad sea menor. En este caso la distribución puede deberse a la necesidad de agrupar varios empleados para desarrollar un mismo proyecto: raramente un DBA, un programador o un analista trabajarán de forma autónoma, sino que se integrarán en un equipo de proyecto. Por el contrario, otros perfiles, como tal vez un investigador o un técnico del área de seguridad, pueden probablemente desarrollar su labor sin necesidad de establecer una colaboración directa de manera prolongada, por lo que, aun siendo conveniente que sean capaces de trabajar en equipo, no se considera imprescindible para dicho puesto.

Finalmente, la Fig. 1(c) muestra que las ofertas que demandan capacidades de liderazgo se corresponden mayoritariamente con puestos de consultor, técnico y arquitecto. Más allá de las observaciones consignadas en la sección anterior respecto al escaso número de ofertas para puestos de responsabilidad y su influencia directa sobre la demanda de la capacidad de liderazgo, la demanda de esta habilidad en puestos técnicos y de consultoría implica que la jerarquía y la asignación de responsabilidades están muy presentes dentro de los equipos de proyecto. La capacidad de liderazgo se antoja imprescindible para sortear posibles complicaciones y/o conflictos que puedan surgir durante el desarrollo de los proyectos y para impulsar el trabajo de los miembros del equipo hacia un objetivo común.

3.4 Relación entre el rango salarial y las habilidades emocionales requeridas

En primer lugar cabe destacar que sólo 310 de las ofertas analizadas (el 24.46% del total de ofertas) incluyen datos salariales, lo que dificulta la obtención de conclusiones al respecto. Esta aparente falta de transparencia se debe a que la mayoría de las empresas prefiere discutir el salario con el candidato o candidatos seleccionados en entrevistas celebradas al final del proceso de selección para evitar que el salario actué como filtro no deseado, ahuyentado potenciales candidatos cuyos perfiles podrían adecuarse a las necesidades de la empresa ofertante.

La Fig. 2(a) ilustra la distribución de ofertas que requieren alguna de las habilidades emocionales en función de los salarios ofertados en cada una.

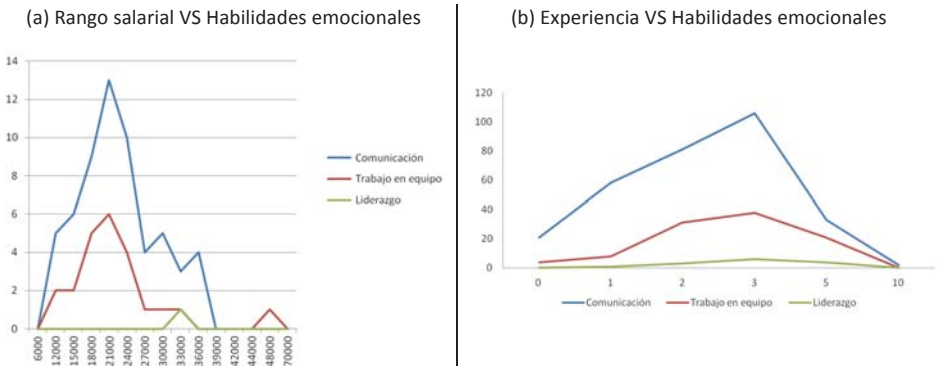


Fig. 2. Relación entre características de la oferta y habilidades emocionales requeridas.

Podría decirse que los datos no son especialmente relevantes a priori, ya que los picos se producen en los rangos salariales *intermedios* (18k-24k €/año), en los que obviamente se incluyen más ofertas y es por tanto mayor la probabilidad de encontrar ofertas que demanden habilidades emocionales.

3.5 Relación entre la experiencia y las habilidades emocionales requeridas

La Fig. 2(b) ilustra la relación entre los años de experiencia que en la oferta se exigen o recomiendan y las habilidades emocionales que se requieren:

En relación a los datos observados en la gráfica, se puede apreciar como a medida que aumenta el número de años de experiencia requeridos, aumenta el número de ofertas que solicitan al menos una de las 3 habilidades emocionales. Esta tendencia se da hasta los 3 años de experiencia. A partir de ese momento, el número de ofertas que demandan habilidades emocionales desciende de manera considerable.

Uno de los motivos por los cuales podría explicarse este punto de inflexión en los 3 años de experiencia sería un incremento en la importancia que los empleadores otorgan a la experiencia que pueda aportar el candidato, frente a cualquier otra competencia. Es decir, el hecho de contar con un número considerable de años de experiencia, se premiaría más que disponer de ciertas capacidades.

Igualmente, en muchos casos se sobreentiende que la experiencia necesariamente implica que se han desarrollado esas capacidades. Por ejemplo, a un jefe de equipo con 5 años de experiencia podemos suponerle capacidades de liderazgo y gestión (nótese que no necesariamente contará con estas capacidades, puesto que podría tratarse de un jefe de equipo que haya ejercido sus responsabilidades de forma inadecuada y/o en ausencia de dichas habilidades).

Finalmente, como en el caso anterior la distribución se ve condicionada por el número de ofertas que caen en los diferentes rangos salariales, dado que el número de ofertas que demandan entre 5 y 10 años de experiencia es sensiblemente menor que el de ofertas que demandan entre 0 y 3 años de experiencia. Ante un número equiparable de ofertas, los resultados podrían ser distintos a los obtenidos.

3.6 Resumen y conclusiones

A continuación se resumen brevemente algunas conclusiones extraídas del análisis realizado sobre la demanda de habilidades emocionales en las ofertas de empleo del sector TIC en general y sobre la relación entre algunas características de dichas ofertas y ciertas habilidades en particular.

Probablemente la conclusión más destacada que se puede extraer de los resultados del análisis es que se puede concluir que efectivamente los empleadores demandan este tipo de habilidades, dado que tal y como muestra la Fig. 3(a), el 81,77% de las ofertas analizadas hace referencia al menos a alguna de las 21 habilidades emocionales consideradas en este estudio.

Cuando se limita el análisis a las 3 habilidades emocionales escogidas como representativas (liderazgo, comunicación y trabajo en equipo), se comprueba que la existen diferencias importantes en la demanda de unas habilidades emocionales frente a otras. Así, la Fig. 3(b) muestra que la capacidad de comunicación es con diferencia la más demandada de las 3.

Fig. 3. (a) Demanda de habilidades emocionales y (b) Distribución de las ofertas que demandan capacidades de comunicación, trabajo en equipo y liderazgo

Podemos intuir que la necesidad de saber comunicarse de una manera eficiente, que siempre ha sido clave en cualquier ámbito, cobra cada día más importancia para los empleadores del sector, dada la creciente tendencia hacia la deslocalización, los equipos geográficamente distribuidos y el trabajo desde casa [1][7].

Si analizamos la distribución por puestos (ver Sección 3.3), se observa que la demanda de habilidades de comunicación y trabajo en equipo se distribuye de una manera bastante equitativa entre las ofertas que seleccionan técnicos, analistas, consultores y programadores, puestos todos ellos que implican interacción y colaboración, mientras que el liderazgo, cuya demanda es sensiblemente menor, predomina en las ofertas de consultor, probablemente porque se espera del consultor que asuma labores de jefe de equipo.

Por otro lado, la falta de datos dificulta analizar la relación entre los salarios ofertados y las habilidades emocionales requeridas, ya que tal como muestra la Fig. 4(a), sólo el 24% de las ofertas incluyen datos al respecto.

Fig. 4. (a) Ofertas de empleo que incluyen datos salariales y (b) Experiencia demandada en las ofertas de empleo analizadas

No obstante, si se ha comprobado que entre aquellas ofertas que si consignan datos salariales, aquellas que demandan la habilidad de comunicación se corresponden con rangos salariales más elevados.

En cuanto a la relación entre la experiencia requerida y las habilidades emocionales demandadas, de las observaciones recogidas en la Sección 3.5 se desprende que las ofertas en las que hay mayor demanda de habilidades emocionales corresponden a requisitos menos exigentes en términos de experiencia

Una observación colateral que cabe destacar es que, tal como muestra la Fig. 4(b), la mayoría de ofertas publicadas solicitan no más de 3 años de experiencia. Probablemente porque las empresas prefieren recurrir a proceso internos de contratación para ocupar puestos de mayor responsabilidad, que tradicionalmente se corresponden con empleados que cuentan con mayor experiencia.

La última reflexión tiene que ver más con una posible limitación del estudio observada al tratar de extraer conclusiones de los datos recogidos (es por ello que se incluye en esta sección) y tiene que ver con la necesidad de *normalizar* los datos para poder derivar conclusiones más precisas. Por ejemplo, de vuelta a la relación entre experiencia requerida y demanda de habilidades, el hecho de que apenas se encuentren ofertas que requieran más de 3 años de experiencia produce un sesgo que habría que paliar, por ejemplo, analizando por separado cada categoría o normalizando las distribuciones de datos. Nótese que un problema similar se detectaba cuando se analizaba la demanda de la capacidad de liderazgo.

No obstante, este replanteamiento del análisis estadístico no debería suponer un inconveniente una vez que disponemos de los datos y sobre todo, de la forma de automatizar su extracción, como se describe en al siguiente sección.

4 Solución tecnológica para el análisis de ofertas de empleo

Esta sección describe los aspectos más relevantes de `VSJobStatsAnalyzer`, la aplicación desarrollada para automatizar la extracción, clasificación y análisis de la información contenida en las ofertas de empleo publicadas en el portal *Infojobs*.

4.1 Descripción funcional

En esencia la aplicación desarrollada se encarga de etiquetar o clasificar las ofertas de empleo recuperadas. Esta clasificación se divide en dos grandes bloques.

Por un lado, se aprovecha la API de *Infojobs* para procesar los datos estructurados de cada oferta publicada y obtener información como el puesto, la experiencia requerida, la categoría, el rango salarial, la ciudad o incluso en enlace a la oferta publicada.

Por otro lado, para procesar los datos no estructurados y recabar información respecto a la demanda de habilidades emocionales, para lo que la API de *Infojobs* no ofrece ninguna funcionalidad, se implementa un proceso de *matching* en base a un conjunto de términos recogidos en varios diccionarios, de forma que si un determinado término se reconoce en el texto de una determinada oferta, ésta será clasificada en la categoría a la que pertenece el término reconocido.

Por ejemplo, se definen 16 términos que se corresponden con la habilidad de comunicación, como 'comunicación', 'don de gentes', 'don de palabra', 'facilidad de palabra', etc., de manera que si una oferta de empleo contiene cualquiera de estos términos se reconoce automáticamente como una oferta que demanda habilidades de comunicación.

Además, para ampliar el ámbito de aplicación de `VSJobStatsAnalyzer`, se han implementado una serie de funcionalidades adicionales que permiten parametrizar las búsquedas de manera que sea susceptible de utilizarse para otros tipos de estudios, como la posibilidad de modificar los diccionarios o la exportación de resultados a hojas de cálculo o ficheros de texto.

4.2 Arquitectura de la herramienta

Esta sección proporciona una vista general de la arquitectura de procesos que soporta la herramienta desarrollada, ilustrada en la Fig. 5.

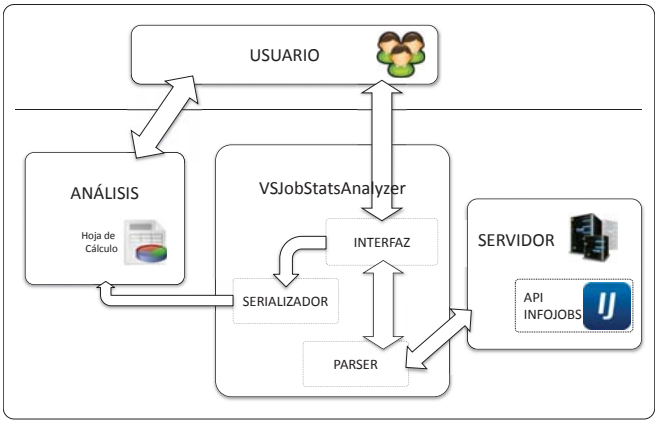


Fig. 5. Arquitectura de procesos de `VSJobStatsAnalyzer`.

A la vista de la figura anterior, se puede apreciar la modularidad del sistema debido a la clara división de los procesos entre sí. Esto permite que en caso de una posible evolución de mejora sobre la aplicación, se pueda realizar de una manera sencilla.

Se distinguen tres componentes principales: el módulo de **Análisis** que se encarga de procesar la información estructurada producida por la aplicación; la aplicación (**`VSJobStatsAnalyzer`**) en sí misma, que se encarga de extraer y clasificar los datos de las ofertas publicadas y finalmente el **Servidor** de *Infojobs* que ofrece una API RESTful que permite extraer información del portal en formatos JSON o XML.

Nótese que esta arquitectura modular simplifica notablemente futuras evoluciones del sistema.

El proceso comienza cuando el usuario fija los criterios de búsqueda utilizando los controles que la aplicación proporciona al efecto, mostrados en la Fig. 6.

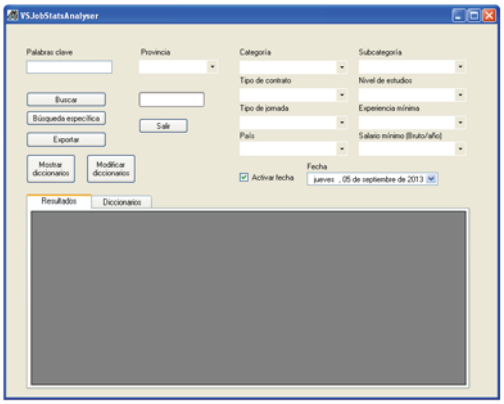
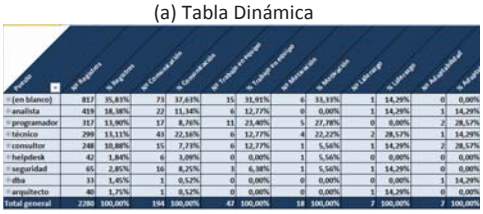


Fig. 6. Interfaz de `VSJobStatsAnalyzer`

A continuación, los datos recuperados, que se muestren en el *Grid* de la parte inferior de la aplicación se exportan a una hoja de cálculo, previa serialización en formato CSV. Finalmente, 3 macros se encargan de utilizar dichos datos para poblar una tabla dinámica que facilita el análisis y la producción de gráficos (la Fig. 7 muestra un par de capturas a modo de ejemplo).

(a) Tabla Dinámica



(b) Gráficos

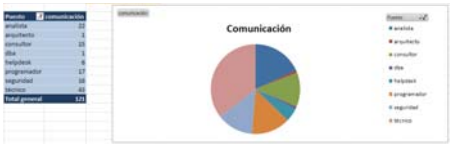


Fig. 7. Capturas de la hoja de cálculo producida por el sistema

5 Conclusiones y trabajos futuros

En este trabajo se ha presentado un estudio sobre la demanda de habilidades emocionales en las ofertas de empleo del sector TIC, para lo que se ha desarrollado

una herramienta que permite automatizar la búsqueda, extracción y clasificación de la información contenida en las ofertas publicadas en el portal *Infojobs*.

Aunque tal y como se señala en el artículo, el estudio puede presentar ciertas limitaciones y admite mejoras en el análisis estadístico de la información, permite corroborar la creciente relevancia que las habilidades emocionales tienen para os empleadores del sector de las TIC (más del 80% de las ofertas de analizadas requerían al menos una de las 21 habilidades emocionales consideradas).

A partir de aquí, se plantean dos direcciones principales para trabajos futuros. Por una parte, ciñéndonos al propio estudio y más allá de las mejoras que admite la aplicación, ampliar el número de fuentes de información, incluyendo no sólo más portales nacionales, sino ampliando el ámbito del estudio a portales europeos o mundiales.

La otra línea de trabajo que ya hemos abordado pasa por estudiar la correlación entre la demanda de competencias del sector empresarial y la formación que reciben los futuros profesionales en la universidad. Así, nos proponemos analizar la presencia y relevancia que tienen las habilidades emocionales en los programas de grado a nivel nacional e internacional y realizar algunas propuestas de mejora orientadas a incluir la enseñanza de estas habilidades en los currículos formativos. De hecho, en el marco de un proyecto local de innovación docente ya hemos desarrollado algunas experiencias que han sido bien acogidas venidas por estudiantes y compañeros [12][13].

Agradecimientos. Esta investigación ha sido llevada a cabo en el marco del proyecto MASAI (TIN-2011-22617), financiado por el Ministerio de Ciencia e Innovación y el proyecto “Aplicación de técnicas de coaching a la enseñanza de habilidades personales”, financiado por la Universidad Rey Juan Carlos.

Referencias

[1] Aspray, W., Mayadas, F., & Vardi, M. Y. (2006). Globalization and offshoring of software. *Report of the ACM Job Migration Task Force, Association for Computing Machinery*.

[2] European Centre for the Development of Vocational Training (Cedefop) and ICEL Career Space, 2002. “Generic ICT skills profiles. Future skills for tomorrow’s world” http://europa.eu.int.

[3] The European Higher Education Area (1999). *Joint declaration of the European Ministers of Education: The Bologna Declaration* (June 19, 1999).

[4] Gardner, H. (1983). *Frames of mind*. New York: Basic Books.

[5] Goleman, D. (1995). Emotional intelligence. Bantam Books.

[6] Goleman, D. (2000). Working with emotional intelligence. Bantam Books.

[7] Herbsleb, J. D., & Moitra, D. (2001). Global software development. *Software, IEEE*, *18*(2), 16-20.

[8] Impagliazzo, J. (2006). Computing curricula 2005. ACM SIGCSE Bulletin, 38(3), 311. ACM and IEEE.

[9] Infoempleo. Informe 2012. http://es.slideshare.net/InfoempleoSlide/presentacin-informe-infoempleo-2012

[10] Ivcevic, Z., Brackett, M. A., & Mayer, J. D. (2007). Emotional intelligence and emotional creativity. Journal of Personality, 75(2), 199-235. Wiley Online Library.

[11] Krueger, F., Barbey, A.K., McCabe, K. Strenziok, M., Zamboni, G., Solomon, J., Raymont. V., and Grafman, J. (2009). The neural bases of key competencies of emotional intelligence. http://www.pnas.org/content/106/52/22486.full

[12] Marcos, E., Vara, J.M., Bollati, V., & López, M. (2014). Applying Coaching Practices to Leadership and Team Management Learning in Computer Science: a Practical Experience. *Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills. A Volume in the Advances in Higher Education and Professional Development (AHEPD)*.18-34. IGI-GLOBAL

[13] Marcos, E., Vara, J. M., Bollati, V., & Lopez, M. (2012, September). Teaching Emotional Intelligence to Computer Science students. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on* (pp. 875-881). IEEE..

[14] Parolia, N., Jiang, J. J., & Klein, G. (2013). The presence and development of competency in IT programs. *Journal of Systems and Software*, *86*(12), 3140–3150.

[15] Payne, W.L. (1983/1986). A study of emotion: developing emotional intelligence; self integration; relating to fear, pain and desire. Dissertation Abstracts International, 47, p. 203A. (University microfilms No. AAC 8605928).

[16] Petrides, K.V., Sangareau, Y., Furnham A., Frederickson, N. (2006). Trait Emotional Intelligence and Children's Peer Relations at School. *Social Development, 15*(3), 537–547.

[17] Suliman, A. M., & Al-Shaikh, F. N. (2007). Emotional intelligence at work: links to conflict and innovation. Employee Relations, 29(2), 208-220.

[18] Thorndike, R. K. (1920). "Intelligence and Its Uses", *Harper's Magazine* 140, 227-335.

[19] Tucker, M.L., Sojka, J.Z., Barone, F.J., McCarthy, A.M. (2000.) Training Tomorrow's Leaders: Enhancing the Emotional Intelligence of Business Graduates. *Journal of Education for Business*, 75(6), 331-337.

[20] Voorhees, R.A. (2001), Competency-Based Learning Models: A Necessary Future. *New Directions for Institutional Research*, 2001(110), 5–13.

Anexo – Habilidades Emocionales

La siguiente tabla describe enumera las 21 habilidades emocionales que se han considerado en este estudio (la selección se basa en las recomendaciones recogidas en [1] y la observación de la información presente en más de 500 ofertas de empleo del sector TIC publicadas en el portal *Infojobs*).

Comunicación	Visión estratégica
Trabajo en equipo	Asertividad
Motivación	Compromiso
Liderazgo	Autonomía
Adaptabilidad	Cualidad de adaptarse
Responsabilidad	Iniciativa
Pensamiento crítico	Resolución de problemas
Creatividad	Resolución de conflictos
Negociación	Empleabilidad
Pro-actividad	Gestión
Trabajo bajo presión	(Capacidad de) Análisis

Una aproximación Ágil a los niveles de madurez 2 y 3 de CMMI-DEV en entornos de desarrollo Web*

C.J. Torrecilla-Salinas[1], J. Sedeño[1, 2], M.J.Escalona[1] and M. Mejías[1]

[1] Grupo IWT2, Universidad de Sevilla
{carlos.torrecilla,jorge.sedeno}@iwt2.org
{mjescalona,risoto}@us.es
[2]Agencia Andaluza de Instituciones Culturales
jorge.sedeno@juntadeandalucia.es

Resumen. Las metodologías y técnicas Ágiles aparecieron hace aproximadamente 10 años y se han convertido en la actualidad en una alternativa para el desarrollo de software. Este tipo de metodologías son especialmente interesantes en el marco de los desarrollos Web, ya que las características de las técnicas Ágiles pueden encajar muy bien con la especificidad propia de un desarrollo Web. Por otro lado, los modelos de madurez como CMMI-DEV, que se centran en la madurez de una organización que desarrolla software, han probado ser mecanismos validos para incrementar la calidad en los procesos de desarrollo. La propuesta de un conjunto de técnicas Ágiles que permita a una determinada organización alcanzar los objetivo específicos de los niveles de madurez 2 y 3 de CMMI-DEV podría ser muy interesante para aquellas organizaciones (sobre todo pequeñas y medianas) que trabajan en proyectos de desarrollo Web, ya que se podría combinar la adaptabilidad proporcionada por las metodologías y técnicas Ágiles, tan necesaria en el entorno Web, con el incremento de madurez en los procesos posibilitado por CMMI-DEV. En este trabajo se propone un conjunto de técnicas, métodos y metodología que, combinados, podrían ayudar a una organización a alcanzar los objetivos genéricos y específicos de los niveles 2 y 3 de CMMI-DEV.

Palabras clave: Metodologías Ágiles, Scrum, Ingeniería Web, CMMI, Ingeniería del Software.

1 Introducción

CMMI-DEV (Capability Maturity Model Integration for Development) forma parte de la familia de modelos CMMI. Estos modelos proporcionan un marco de comparación que permite evaluar la madurez de una organización a la hora de desarrollar software [1], asumiéndose que alcanzar sus niveles de madurez más altos está relacionado directamente con incrementos en la calidad del software [2]. Por otro lado, los desarrollos de software en entornos Web tienen ciertas características especiales, co-

* *Las opiniones expresadas en este artículo son las de sus autores, no siendo necesariamente compartidas por sus empleadores*

mo una aproximación flexible a la gestión de requisitos y la necesidad de ciclos cortos de realimentación con los usuarios [3]. En este tipo de desarrollos se pueden encontrar características como la reducción del "time-to-market", la adaptación rápida a requisitos poco definidos o la entrega rápida de valor [4]. Como se sabe, uno de los principios de las metodologías Ágiles es el abrazar el cambio [5], el cual puede encajar muy bien en el marco de los proyectos de desarrollo Web. Sin embargo, a medida que las aplicaciones Web se vuelven más populares también aumenta sus requisitos de calidad. Si, como se ha mencionado, los niveles más altos de CMMI-DEV están relacionados con mejoras en la calidad; el uso de métodos Ágiles para alcanzar los objetivos propuestos por estos niveles de madurez podría ofrecer a las organizaciones que desarrollan sistemas en entornos Web la posibilidad de combinar calidad y madurez mientras son capaces de responder rápidamente al cambio. En particular, los niveles 2 y 3 de madurez de CMMI-DEV podrían representar un buen compromiso entre nivel de formalismo y adaptabilidad [6, 7]. En base a lo expuesto, el presente trabajo tiene como objetivos: Presentar una serie de metodologías, técnicas y métodos Ágiles que puedan cumplir los objetivos de los niveles de madurez 2 y 3 de CMMI-DEV en entornos Web, proponer un mapeo entre este conjunto frente a los citados objetivos, y extraer una serie de conclusiones relevantes e identificar líneas de investigación.

Este trabajo está organizado en los siguientes apartados. Tras esta introducción , el Apartado 2 presenta los trabajos relacionados. El Apartado 3 plantea el conjunto de métodos Ágiles propuesto y el Apartado 4 el mapeo entre este conjunto y los objetivos específicos de los niveles 2 y 3 de CMMI. Por último, el Apartado 5 extrae las principales conclusiones y plantea líneas futuras de investigación.

2 Trabajos relacionados

En este apartado se va a presentar un análisis de los trabajos relacionados existentes, entendiendo éstos como aquellos que, bien partiendo del mundo Ágil, bien partiendo del mundo CMMI, intentan incorporar valores y prácticas del otro campo. Así, podemos encontrar trabajos como los de Jakobsen *et al.* [8, 9] o Sutherland *et al.* [10], que describen, en forma de casos de estudio, como organizaciones con una fuerte cultura CMMI han incorporado prácticas Ágiles en sus procesos de desarrollo. Sin embargo estos trabajos presentan, en general, a organizaciones ya certificadas en CMMI con lo que ni describen como una organización podría avanzar a través de los distintos niveles de madurez usando exclusivamente prácticas Ágiles, ni presentan un mapeo entre las diferentes técnicas Ágiles utilizadas y los objetivos de CMMI, ni se centran en entornos Web. En el trabajo de Lina *et al.* [11] encontramos un análisis de cual aproximación sería más adecuada para alcanzar los objetivos de CMMI, si la basada en Scrum o una más formalista. Sin embargo, las conclusiones que se presentan no son exhaustivas, ya que no se analizan gran parte de las áreas de proceso, además de centrarse únicamente en Scrum como método Ágil. Por otro lado podemos encontrar trabajos como los de Cohan *et al.* [12], Garzas et *al.* [13] y Baker [14, 15], en los que, también en forma de casos de estudio, se presentan ejemplos de empresas que, usando Scrum o XP, han sido capaces de pasar con éxito evaluaciones CMMI. En el caso del

estudio de Garzas, se analiza la viabilidad del uso de Scrum para cubrir los objetivos del nivel 2 de CMMI, sin mencionar los objetivos del nivel 3. Los trabajos de Cohan y Baker se centran más en la descripción del proceso de evaluación, sin proporcionar un mapeo entre las diferentes prácticas y objetivos, o bien haciéndolo a muy alto nivel. Por último, todos estos trabajos se centran en proyectos de desarrollo de software generalistas, sin tener en cuenta las especificidades de los desarrollos Web. Dentro de los trabajos relacionados podemos encontrar también un conjunto de trabajos teóricos, como los de Lukasiewicz *et al.* [6], Marçal *et al.* [7] y Diaz *et al.* [16]. Estos trabajos presentan un mapeo entre un cierto conjunto de prácticas Ágiles y los objetivos de un cierto número de áreas de proceso de CMMI en los niveles 2 y 3. En general, en estos trabajos, encontramos que no hay un mapeo completo de todos los objetivos de todas las áreas de proceso (centrándose únicamente en algunas de ellas) y que se centran en procesos de desarrollo genéricos, sin tener en cuenta las características específicas de los proyectos Web. Como conclusión, podemos observar que no existen trabajos que propongan un mapeo completo entre un conjunto de técnicas Ágiles y todos los objetivos específicos de todas las áreas de proceso de los niveles de madurez 2 y 3 de CMMI-DEV en entornos de desarrollo Web. Sí podemos encontrar en nuestro trabajo previo [17] un mapeo entre Scrum y todas las áreas de proceso del nivel 2 de CMMI-DEV, identificando los "gaps" y proponiendo técnicas Ágiles para cubrir los mismos. Éste es el único trabajo que se centra en procesos de desarrollo Web y lo podemos considerar el punto de partida de este estudio. Partiendo de él, presentaremos un conjunto de técnicas Ágiles que puedan cubrir todos los objetivos de las áreas de proceso de los niveles 2 y 3 de CMMI-DEV para proyectos de desarrollo Web.

3 Presentación del conjunto de técnicas Ágiles

Nuestra propuesta de técnicas Ágiles se basa en el *framework* Scrum, que es la metodología Ágil más popular [18]. Alrededor de Scrum se propone tanto un conjunto de metodologías, técnicas y métodos Ágiles y como extensiones ad-hoc al mismo. Las metodologías, métodos y técnicas Ágiles adicionales se presentan en la Tabla 1 y las extensiones ad-hoc a Scrum propuestas se presentan en la Tabla 2:

Tabla 1. Metodologías, métodos y técnicas Ágiles propuestas.

Propuesta	Descripción
Scrum	Se propone el uso de Scrum [19] como marco para la gestión del proyecto y sus requisitos. Permitirá adecuarse a la gestión de requisitos difusos propia de desarrollos Web, así como reducir los ciclos de realimentación por parte de los usuarios.
XP	Del conjunto de técnicas propuestas por eXtreme Programming [20], se propone incorporar las siguientes: Propiedad compartida del código, Control de versiones, Integración continua y Programación en parejas. Estas técnicas permiten alinearse con la necesidad de continuidad de negocio, reducción de "time-to-market" y entrega rápida que, cómo se ha mencionado, caracterizan a los desarrollos Web.
Gestión Ágil de Riesgos	La gestión Ágil de riesgos [21] es una técnica que permite integrar en el ciclo de vida de Scrum la revision y evaluación continua de los riesgos, así como su reporte a la dirección, lo que se alinea con la necesidad de ciclos de realimentación cortos de los desarrollos Web.
Contratación Ágil	La contratación Ágil [22] es una técnica que permite crear una relación colaborativa entre cliente y proveedor, facilitando la adaptación a cambios en requisitos característica de los desarrollos Web.

Lean	En relación a Lean, se propone la inclusión de técnicas como grupos de trabajo para la mejora [23] o amplificación del conocimiento [24], que permitirán alinear los procesos de mejora y de formación sistemática con los ciclos de realimentación cortos necesarios para los desarrollos Web
Agile Project Management	Se propone incorporar la fase inicial de visionado propuesta por Agile Project Management [25], así como sus diferentes técnicas de toma de decisiones, lo que permitirá ayudar a la toma de decisiones incluso en situaciones en las que las opciones no están del todo definidas.
Pruebas Ágiles	Se propone la inclusión de dos técnicas: Test Driven Development (TDD) [26] para las pruebas unitarias y Acceptance Test Driven Development (ATDD) [27] para las pruebas de aceptación o de usuario. Esta aproximación permitirá una definición temprana de los tests, incorporando al proceso de prueba a los propios usuarios.
Arquitectura Ágil	Las técnicas de arquitectura Ágil [28], incluyendo conceptos como diseño emergente, "testing" exploratorio, diseño y arquitectura emergente permitirán adecuar el diseño a unos requisitos inicialmente difusos que se van definiendo a lo largo del proyecto.

Tabla 2. Extensiones a Scrum.

Extensión	Descripción
Scrum a nivel de empresa	La implementación de Scrum a nivel de empresa propuesta por Schwaber [29] permitirá la definición de Scrum como un proceso a nivel de la organización, y su adaptación a las características de la organización y los proyectos
Sprint 0	Tal y como se incluye en nuestro trabajo previo [17], se propone la realización de un Sprint inicial de planificación (también denominado Sprint 0), que permitirá establecer las bases de la gestión y el desarrollo posterior del proyecto.
Otras extensiones	Estas extensiones a Scrum, propuestas en nuestro anterior trabajo [17], son relativas a la medición de indicadores de rendimiento, gestión y distribución de información, reports, a la definición de indicadores de calidad y su gestión…

4 Propuesta de mapeo

La Tabla 3 muestra la propuesta de mapeo entre las áreas de proceso de los niveles 2 y 3 y el conjunto de metodologías, métodos y técnicas propuesto, basada en el análisis del cumplimiento de los objetivos de las áreas de proceso de CMMI-DEV. El modelo que se plantea debe ser formalizado e integrado de forma consistente así como evaluado, a través de una evaluación formal o autoevaluación, aspectos que se plantearán como líneas de investigación futuras:

Tabla 3. Propuesta de mapeo.

	Áreas de proceso Nivel 2							**Áreas de proceso Nivel 3**									
	CM	**MA**	**PMC**	**PP**	**PPQA**	**REQM**	**SAM**	**DAR**	**IPM**	**OPD**	**OPF**	**OT**	**PI**	**RD**	**RSKM**	**TS**	**VAL**	**VER**
Propuesta	XP	Scrum y Extensiones				Scrum	Contratos Ágiles	Agile Project Management	Scrum a nivel empresa		Lean		XP	Scrum	Gestión Ágil de Riesgos	Arquitectura Ágil	ATDD	XP

5 Conclusiones y trabajos futuros

La conclusión principal de nuestro trabajo es que la propuesta de un conjunto de técnicas Ágiles permitir la consecución de todos los objetivos específicos de los niveles 2 y 3 de madurez de CMMI-DEV para entornos de desarrollo Web. Como líneas de trabajo futuras se puede sugerir la formalización e integración de las técnicas en un *framework* consistente, así como continuar con la identificación de técnicas, prácticas y metodología Ágiles que, de forma conjunta, pudieran permitir a las organizaciones alcanzar el resto de niveles de madurez de CMMI-DEV (4 y 5). La definición y formalización de este conjunto de metodologías, técnicas y prácticas Ágiles, combinado con el conjunto ya identificado, permitiría a las organizaciones que desarrollan sistemas Web progresar a través de los diferentes niveles de madurez de CMMI, manteniendo a un nivel de formalismo adecuado y la capacidad de adaptarse rápidamente a los cambios, mientras que el proceso de desarrollo Ágil se institucionaliza y la calidad de los desarrollos aumenta. Combinar esta definición con el desarrollo de herramientas de soporte que puedan ser utilizadas por las organizaciones sería otra línea de trabajo. Para finalizar, una validación del modelo propuesto en una implementación real a través de una evaluación formal o una auto-evaluación será también necesaria, de cara a validar las propuestas incluidas en este trabajo.

6 Agradecimientos

Esta investigación ha sido apoyada por el proyecto MeGUS (TIN2013-46928-C3-3-R) del Ministerio de Ciencia e Innovación, el proyecto Tempros (TIN2010-20057-C03-02), y el proyecto NDTQ-Framework (TIC-5789) de la Junta de Andalucía.

Referencias

1. CMMI Product Team. “CMMI for Development, Version 1.3.” Nov. 2010. Carnegie Mellon University. http://www.sei.cmu.edu/reports/10tr033.pdf. Accedido marzo 2014.
2. Goldenson, D. et al. “Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results”. Carnegie Mellon University, 2003. http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=6365. Accedido marzo 2014.
3. Ran, H. et al. “Agile Web Development with Web Framework”. Actas de 4th Int. Conf. on Wireless Communications, Networking and Mobile Computing. (Dalian, China, 12-17 Oct. 2008) WiCOM '08. IEEE.
4. Pressman, R.S. “What a Tangled Web We Weave”. *IEEE Software*, Jan.-Feb., pp. 18-21. Jan-Feb. 2000.
5. Beck, K. et al. “Manifesto for Agile Software Development”. 2001. http://www.agilemanifesto.org. Accedido marzo 2014.
6. Lukasiewicz, K. et al. "Improving Agility and Discipline of Software Development with the Scrum and CMMI," Software, IET, vol.6, no.5, pp.416-422, October 2012.
7. Marcal, A.S.C et al. “Blending Scrum Practices and CMMI Project Management Process Areas”. 2008. ISSE 4. 17-29.

8. Jakobsen, C.R.; Sutherland, J. “Scrum and CMMI Going from Good to Great”. Actas de Agile Conference 2009 (Chicago, IL, USA 24-28 Aug. 2009). AGILE ‘09. IEEE.
9. Jakobsen, C. R.; Johnson, K. A. “Mature Agile With a Twist of CMMI”. Aug. 2008. Actas de Agile Conference 2008 (Toronto, Canada 04-08 Aug. 2008). AGILE ‘08. IEEE.
10. Sutherland, J. et al. “Scrum and CMMI Level 5: The Magic Potion for Code Warriors”. Jan. 2008. Actas de 41st annual Hawaii Int. Conf. on System Sciences 2008 (Hawaii, USA 07-10 Jan. 2008). HICSS ’08. IEEE.
11. Lina, Z. et al. "Research on Combining Scrum with CMMI in Small and Medium Organizations". Actas de 2012 Int. Conf. on Computer Science and Electronics Engineering (Hangzhou, China 23-25 Mar. 2012). ICCSEE ’12. IEEE.
12. Cohan, S. et al. “An Agile Development Team’s Quest for CMMI Maturity Level 5”. Actas de Agile Conference 2009 (Chicago, USA 24-28 Aug. 2009). AGILE ‘09. IEEE.
13. Garzás, J. et al. “A case study of software process improvement with CMMI-DEV and Scrum in Spanish companies”. Journal of Software: Evolution and Process, vol. 25, no.12, pp.1325-1333. 2013.
14. Baker, S. W. “Formalizing Agility, Part 2: How an Agile Organization Embraced the CMMI”. Jul. 2006. Actas de Agile Conference 2006 (Minneapolis, USA 23-28 July 2006). AGILE ‘06. IEEE.
15. Baker, S. W. “Formalizing Agility: An Agile Organization’s Journey Toward CMMI Accreditation”. Jul. 2005. Actas de Agile Development Conference 2005 (Denver, USA 24-29 July 2005). ADC ‘05. IEEE.
16. Diaz, J. et al. “Mapping CMMI Level 2 to Scrum Practices: An Experience Report”. Jan. 2009. SPI 42 93-104.
17. Torrecilla-Salinas, C. et al. “A Scrum-based Approach to CMMI Maturity Level 2 in Web Development Environments”. Actas de Int. Conf. on Information Integration and Web-based Applications & Services 2012 (Bali, Indonesia 3-5 Dec. 2012). iiWAS, 12. ACM.
18. Pikkarainen, M. et al. “The Impact of Agile Practices on Communication in Software Development”. Empirical Softw. Engg. Springer, pp 303-337. May. 2008.
19. Sutherland, J.; Schwaber, K. “The Scrum Guide: The Definitive Guide to Scrum”. 2011. http://www.scrum.org/Scrum-Guides. Accedido en marzo de 2014.
20. Beck, K.; Andres, C. “Extreme Programming Explained: Embrace Change, Second Edition”. Boston: Addison-Wesley. 2004.
21. Lant, M. “Five Simple Steps to Agile Risk Management”. From http://michaellant.com/2010/06/04/five-simple-steps-to-agile-risk-management/. Accedido marzo 2014
22. Jamieson, D. et al. "Agile Procurement and Dynamic Value for Money to Facilitate Agile Software Projects," Sept. 2006. Actas de 32nd EUROMICRO Conf. on Software Engineering and Advanced Applications (Dubrovnik, Croatia, Aug. 29-Sept. 1 2006). SEAA '06.
23. Kniberg, H. “Lean from the Trenches: Managing Large-Scale Projects with Kanban”. Dallas: The Pragmatic Bookshelf. 2012.
24. Poppendieck, M.; Poppendieck, T. “Lean Software Development. An Agile Toolkit”. Boston: Addison-Wesley. 2003.
25. Highsmith, J. “Agile Project Management: Creating Innovative Products, Second Edition”. NJ: Addison-Wesley. 2009.
26. Beck, K. “Test Driven Development. By Example”. Boston: Addison-Wesley. 2008.
27. Adzik, G. “Specification by Example: How Successful Teams Deliver the Right Software”. CT: Manning Publications. 2011.
28. Shore, J.; Warden, S. “The Art of Agile Development”. CA: O’ Reilly. 2007.
29. Schwaber, K. “The Enterprise and Scrum”. Redmond: Microsoft Press. 2007.

Usability through Software Design

Laura Carvajal*, Ana M. Moreno*, María-Isabel Sánchez-Segura^, Ahmed Seffah˜
* Universidad Politecnica de Madrid, Spain
^ Universidad Carlos III de Madrid, Spain
˜ Concordia University, Canada
{ammoreno@fi.upm.es, lauraelena.carvajal@upm.es, misanche@inf.uc3m.es,
seffah.ahmed@concordia.edu}

Abstract. Over the past two decades the HCI community has proposed specific features that software applications should include to overcome some of the most common usability problems. However, incorporating such usability features into software applications may not be a straightforward process for software developers who have not been trained in usability (i.e., determining when, how, and why usability features should been considered). We have defined a set of usability guidelines for software development to help software engineers incorporate particular usability features into their applications. In this paper we focus on the software design artifacts provided by the guidelines. We detail the structure of the proposed design artifacts and how they should be used according to the software development process and software architecture used in each application. We have tested our guidelines in an academic setting. Preliminary validation shows that the use of the guidelines reduces development time, improves the quality of the resulting designs and significantly decreases the perceived complexity of the usability features from the developers' perspective.

1 Motivation

Usability is defined by ISO as "the capability of a software product to be understood, learned, used and attractive to the user, when used under specified conditions". This attribute has gained importance as an integral quality aspect of software development.

For over a decade, the software engineering (SE) community has been actively pursuing different lines of research targeting the incorporation of usability practices into software development. Although many valuable attempts have been made to bridge the gap between usability and software design, the problem is still an open research topic.

Our objective is to put another brick in the wall by defining guidelines to help developers build specific usability characteristics into their applications. We have named these guidelines usability guidelines for software development.
We hypothesize that applying the proposed guidelines will: reduce development time; reduce perceived usability-related functionality complexity for developers; improve the quality of resulting software designs.

This paper details the empirical validation of these hypotheses while discussing the proposed guidelines for specific usability features, which we have termed functional usability features (usability features with major impact on software design).

2 Main contributions

As shown in Fig. 1, each guideline is comprised of seven artifacts, distributed as analysis and design artifacts. However, as discussed in the paper, developers can use these artifacts at their convenience, irrespective of their classification into the analysis or design categories.

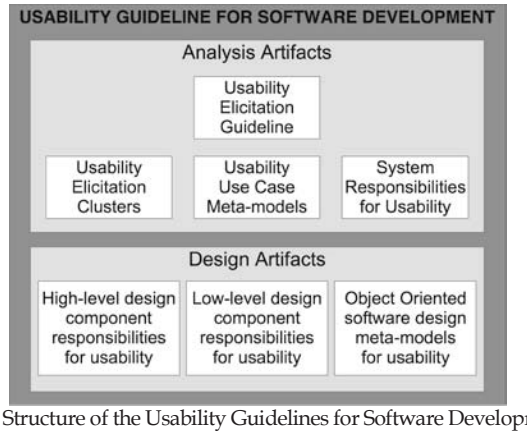


Fig. 1. Structure of the Usability Guidelines for Software Development

Preliminary validation results show a significant decrease in development time; developers who applied the guidelines built their designs more quickly. Furthermore, applying the guidelines also reduced testing time and helped developers produce enhanced designs, especially designs with better responsibility allocation. Finally, the proposed guidelines helped developers perceive usability features as not being overly complex. This finding is crucial as this perception could ultimately predispose a development team towards or against usability.

3 Reference

Title: Usability through Software Design.

Authors: Laura Carvajal, Ana Moreno, Maria-Isabel Sanchez-Segura, Ahmed Seffah.

Publication: IEEE Transactions on Software Engineering, Vol. 39 (11), pp:1582-1596. Noviembre 2013. DOI: 10.1109/TSE.2013.29.

A fine-grained analysis of the support provided by UML class diagrams and ER diagrams during data model maintenance

Gabriele Bavota[1], Carmine Gravino[1], Rocco Oliveto[2], Andrea De Lucia[1], Genoveffa Tortora[1], Marcela Genero[3], and José A. Cruz-Lemus[3]

[1]University of Salerno, Italy
{gbavota, gravino, adelucia, tortora}@unisa.it
[2]University of Molise, Italy
rocco.oliveto@unimol.it
[3]University of Castilla-La Mancha, Spain
{Marcela.Genero, JoseAntonio.Cruz}@uclm.es

Abstract. This paper presents the results of an empirical study aiming at comparing the support provided by ER and UML class diagrams during maintenance of data models. We performed one controlled experiment and two replications that focused on comprehension activities and another controlled experiment on modification activities related to the implementation of given change requests. The results achieved were analyzed at a fine-grained level aiming at comparing the support given by each single building block of the two notations. Such an analysis is used to identify weaknesses (i.e., building blocks not easy to comprehend) in a notation and/or can justify the need of preferring ER or UML for data modeling. The analysis revealed that the UML class diagrams generally provided a better support for both comprehension and modification activities performed on data models as compared to ER diagrams. Nevertheless, the former has some weaknesses related to three building blocks, i.e., multi-value attribute, composite attribute, and weak entity. These findings suggest that an extension of UML class diagrams should be considered to overcome these weaknesses and improve the support provided by UML class diagrams during maintenance of data models.

Keywords: Model maintenance, UML class diagrams, ER diagrams, Comprehension, Family of experiments, Meta-analysis.

1 Publication summary

Understanding and interpreting data models represents a fundamental activity from the earliest stages of software development, e.g., requirement analysis. Thus, a comprehensive notation is really desirable to avoid misunderstanding that can lead to the introduction of errors very expensive to remove in the later phases of the software development.

A comprehensive notation is also desirable to facilitate the maintenance of the produced data model. Indeed, data models should be considered living documents that will change in response to a changing business. Thus, a comprehensive notation can facilitate the comprehension activities that have to be performed to understand the data model before the analysis and the implementation of a change request.

A fine-grained analysis is vital to provide insight on why UML class diagrams are better than the ER diagrams or vice versa and to highlight strengths and limitations of the two notations. This kind of analysis can be used to (i) justify the need of preferring ER or UML class diagrams for data modeling, or (ii) identify weaknesses in a notation that could be overcome to improve its support for data modeling.

In this paper, we present the results of a set of controlled experiments to deeply analyze the support given by ER and UML class diagrams during maintenance of data models. In particular, we conducted a controlled experiment and two replications aiming at analyzing the support given by the two notations during comprehension tasks. In addition, we conducted another controlled experiment to analyze the support given during modification activities. We focus the attention only on these two activities, since high comprehensibility and high modifiability are the two typical expectations of people using these two notations to represent conceptual data models.

The experiments aimed at performing a fine-grained analysis to (objectively and subjectively) compare the single building blocks, i.e., Entity, Primary Key/ID, Composite Attribute, Multi-value Attribute, Recursive relationship, Relationship cardinality, Ternary relationship, Generalization IS-A, Weak entity, M:N relationship, of the two notations. They also aimed at analyzing the comprehensibility of the two notations involved 156 students of the University of Salerno (Italy) with different academic background represented by fresher, bachelor, and master's students. The experiment conducted to analyze the modifiability of the two notations involved 28 master's students of the University of Salerno (Italy).

The results achieved indicate that, regarding to the comprehension activities, the UML class diagrams are characterized by three weaknesses related to the representation of Composite attribute, Multi-value attribute, and Weak entity, as compared to the ER diagrams. However, except for the three identified weaknesses, the UML class diagrams are generally more comprehensible than the ER diagrams. On the other hand, regarding the modification activities, through an objective analysis, two weaknesses of the UML class diagrams were statistically identified, i.e., Multi-value attribute and Weak entity. However, a subjective analysis confirmed also the third weakness identified for the comprehension activities, i.e., the Composite attribute. Moreover, the UML class diagrams generally provide a better support than the ER diagrams also for the modification activities, because with UML class diagrams, on six out of the ten building blocks, students achieve better results.

As it always happens with empirical studies, replications in different contexts, with different subjects and objects, is the only way to corroborate our findings. It would be interesting to consider alternative experimental settings in several respects, but maybe the most important one is the profile of the involved subjects. Replicating this study with students/professionals having a different background would be extremely important to understand how much our findings can be generalized.

EXEMPLAR: An Experimental Information Repository for Software Engineering Research

Jose Antonio Parejo, Sergio Segura, Pablo Fernandez, and Antonio Ruiz Cortés

University of Sevilla, Spain
ETSII. Avda. de la Reina Mercedes s/n, 41012, Sevilla.
japarejo@us.es

Abstract. The number and variety of experiments carried in software engineering research is growing, leading to a increasing need of replication and review. In order to support such needs the information about experiments should be provided as lab-packs comprising of: a description of the experiment, the materials used and data generated during the conduction, and the results of the analyzes performed on such data. However, this information is often scattered, poorly structured, and even unavailable, implying a tedious process of search and gathering. EXEMPLAR is an online platform for managing experimental information, that allows the uploading and publication of experimental lab packs, and an efficient search. The platform also supports the use of formal languages for providing experimental descriptions (e.g. SEDL). In so doing, EXEMPLAR enables the automated analysis of lab-packs, in order to detect common validity threats and missing information which could hinder replicability.

Keywords: empirical research, experiments, experimental replicability, experimental repositories

1 Introduction

In science, the quality of an experiment is determined primarily by two factors: its degree of validity and its replicability. According to [1], "*The use of precise, repeatable experiments is the hallmark of a mature scientific or engineering discipline*".

EXEMPLAR (EXpErimets Management PLAtfoRm) is an online repository of experimental information that aims at supporting the creation of high quality experiments. EXEMPLAR focuses on easing the publication of lab-packs, supporting efficient search, and assuring the quality of experimental information through the automated detection of common validity threats and missing information (based on the analysis of experimental descriptions when provided in formal languages such as SEDL [2] and on the analysis of lab-packs contents). In the next section the main features of EXEMPLAR are described. The repository is available for public access at http://exemplar.us.es.

2 EXEMPLAR features

2.1 Experimental information repository

EXEMPLAR is essentially an online repository of experimental information that provides: i) support for uploading and controlling the availability of experimental information, through the creation of workspaces that can contain several labpacks; and ii) support for searching experiments based on keywords, several classification taxonomies, and on the indexation of the contents of the labpacks. Those features are described in detail below.

Information Organization: Workspaces, labpacks and access control. In EXEMPLAR, each registered user has its own personal space with a maximum quota (currently limited to 1Gb). In such personal space, users can create an unlimited number of workspaces, for which they can control access. Currently workspaces are either public (meaning availability to anyone in read-only mode) or private, but authors plan to implement workspace sharing among users in read-write mode as future work. Each workspace contains an unlimited number of lab-packs. Each lab-pack contains the information of a single experiment, structured as an unlimited number of files and nested folders. The workspaces and labpacks management interface of EXEMPLAR is shown in figure 1 (left side).

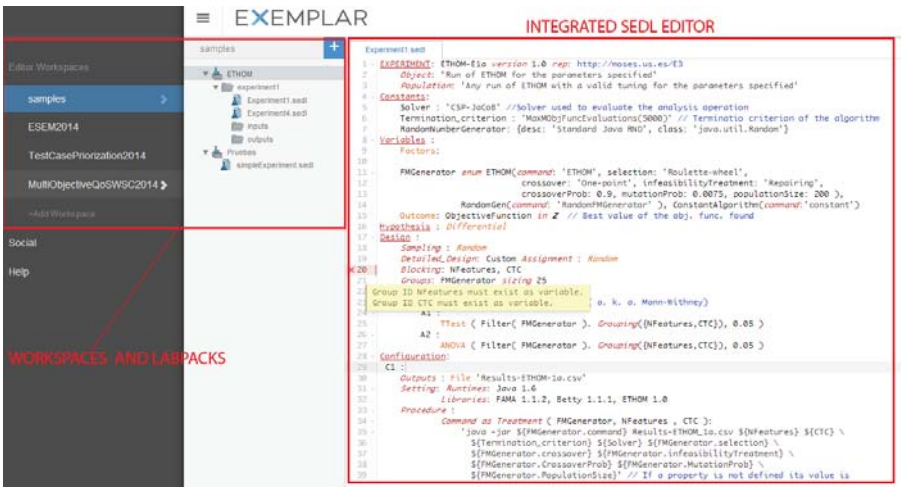


Fig. 1. EXEMPLAR interface for workspace and lab-packs management

Search and Indexation. EXEMPLAR supports two different search mechanisms. On the one hand, users can search labpacks based on their tags or classification according to standard taxonomies (for software engineering experiments we support the SWEBOK taxonomy of areas [1]). On the other hand,

users can perform full text searches on the indexed contents of the labpacks. Figure 2 shows the search page of EXEMPLAR.

Fig. 2. Experimental information search in EXEMPLAR

2.2 Support for experimental description and lab-packs layout

Apart from its capabilities as information repository, EXEMPLAR supports the creation formal descriptions of experiments based on SEDL [2], and it provides a default lab-pack directory layout for inducing a tidy structure on lab-packs contents.

Formal experimental descriptions with SEDL. Formal description of experimental information in EXEMPLAR is supported through an integrated SEDL editor. SEDL (Scientific Experiment Description Language) is a generic language to describe experiments in a precise, unambiguous and tool-independent way. SEDL documents include the information that any experimental description should provide regardless of the application domain: objects, subjects, population, variables, hypothesis, treatments and analysis to be performed. A detailed description of the syntax of SEDL along with several examples are provided in [2]. Figure 1 shows that integrated editor support **syntax coloring**, **sections code folding**, **auto-save** and **error highlighting as you type**.

Layout of experimental information (according to SEA). Experimental reproducibility requires not only providing a comprehensive and detailed description, but also providing all the input and output data of the experiment,and any experimental artifact used for its conduction, such as survey forms, data gathering spreadsheets, etc. The role of those elements in the experiment should have an impact on their location on the lab-pack, in order to ease its use. Thus, a generic default layout for the elements of lab-packs named SEA (Scientific Experiments direcory lAyout) is described in detail in [2, appendix E]. EXEMPLAR supports the creation of SEA-compliant lab-packs, providing advices on where the uploaded files should be located depending on the specific extension and role for such workspaces (identified by asking users).

2.3 Automated analysis of experimental information

The killer feature of EXEMPLAR is its support for the automated analysis of experimental information. This feature enables the detection of: i) common validity threats in the experiments, and ii) inconsistencies between SEDL experimental descriptions and the actual contents of the lab-pack.

Automated detection of common validity threats. Currently, EXEMPLAR supports three operations that can detect common validity threats identified in the literature:

Multiple Comparison: This operations checks if a single comparison statistical test is being used to perform multiple comparisons, leading to a statistical analysis validity threat [3].

Small sampling: This operation checks if the sample size of the experiment is sufficient for a safe application of the statistical tests specified for the analysis in its SEDL description, leading to a statistical analysis validity threat [3]. Currently, this operation checks a minimum sample size of 30 observations in null hypothesis statistical tests are applied.

Inconsistent Variable Measurement: This operation checks if the values of the experimental variables contained in the inputs and output datasets provided, are consistent with the corresponding domains specified in the SEDL description. Depending on the type of data inconsistency (missing value, value out of variable domain), and dataset role (input or output), this inconsistency can be symptomatic of a specific validity threat [2]. Authors plan to implement the whole catalog of analysis operations described in [2] as future work.

Automated detection of missing experimental information in lab-packs. EXEMPLAR supports the checking of the consistencies between inputs, and outputs specified in SEDL experimental descriptions and the actual contents of the corresponding lab-pack. In this sense the repository can remind users to upload forgotten files, or find errors in the inputoutput files specification of SEDL descriptions.

Acknowledgments

This work was partially supported by the EU Commission (FEDER), the Spanish and the Andalusian R&D&I programme grants TAPAS (TIN2012–32273), COPAS (P12–TIC–1867), and THEOS (TIC–5906).

References

1. IEEE Computer Society: Software Engineering Body of Knowledge (SWEBOK). Angela Burgess, EUA (2004), http://www.swebok.org/
2. Parejo, J.A.: MOSES: a Metaheuristic Optimization Software Ecosystem. Ph.D. thesis, Univ. of Sevilla (2013), http://www.isa.us.es/publications
3. Shadish, W.R., Cook, T.D., Campbell, D.T.: Experimental and quasi-experimental designs for generalized causal inference. Houghton Mifflin, 2 edn. (2001)

Suite de aplicaciones colaborativas para dar soporte a la gamificación del prototipado de procesos

Manuel Trinidad y Mercedes Ruiz

Grupo de Investigación en Mejora del Proceso Software y Métodos Formales
Departamento de Ingeniería Informática
Universidad de Cádiz
C/ Chile, 1. 11003 – Cádiz
{manuel.trinidad, mercedes.ruiz}@uca.es

Resumen. Este artículo presenta tres aplicaciones denominadas SysDyn, OpenBadgesUCA y GamAnalyze, que pretenden satisfacer un objetivo común: dar soporte a la aplicación de estrategias de gamificación en el ámbito del prototipado de negocios. Por un lado, SysDyn, cubre la ausencia de aplicaciones de prototipado de negocios basadas en simulación para dispositivos móviles tales como tablets. Por otro, OpenBadgesUCA se encarga de actuar como intermediario para poder subir las medallas logradas por los usuarios de SysDyn a la plataforma de almacenamiento de medallas Mozilla OpenBadges y de introducir los datos pertinentes de la experiencia en una base de datos. Finalmente, GamAnalyze se encarga de analizar la información relacionada con la estrategia de gamificación, procedente tanto del uso de SysDyn como de otras posibles aplicaciones registradas en OpenBadgesUCA, a fin de ayudar en la valoración de los resultados de dicha estrategia.

Palabras clave: Prototipado de negocio, gamificación, simulación, toma de decisiones.

1 Introducción

Actualmente, en cualquier sector productivo no se toma ninguna decisión sin el diseño de planes detallados ni la construcción de prototipos a escala que permitan anticipar los riesgos o el efecto de las decisiones tomadas. Sin embargo, no es frecuente encontrar en las organizaciones software estos modelos que vayan más allá de una representación simplificada de la realidad en una hoja de cálculo.

Un prototipo es un modelo temprano que se utiliza para probar una idea o un proceso a fin de poder aprender de su uso y proporcionar las especificaciones para implementar dicha idea o proceso en el mundo real. El prototipado de procesos es la acción de experimentar con nuevos conceptos en un entorno libre de riesgos. Su objetivo fundamental es explorar las consecuencias que tendrían nuevas ideas, estrategias y diseños antes de llevarlas a la práctica en el negocio real [2].

Por otro lado, en el ámbito de la mejora de procesos, la motivación y el compromiso de los miembros del equipo son elementos indispensables para el éxito de la creación y puesta en marcha de iniciativas. Por ello, siempre es importante aplicar medi-

das enfocadas a fomentar el compromiso y la implicación de quienes deben idear y ejecutar las medidas de mejora de los procesos.

En los últimos años, la gamificación ha cobrado una especial atención por parte de diferentes sectores como elemento que contribuye a aumentar los niveles de motivación y de implicación de las personas. La gamificación, definida como la aplicación de mecánicas y dinámicas de juego en entornos no lúdicos, permite enriquecer una experiencia, contribuyendo a aumentar la motivación y a fomentar determinados comportamientos [1].

En este trabajo se presenta la primera versión de tres aplicaciones colaborativas que persiguen crear un entorno donde los equipos de desarrollo de software se vean motivados hacia el diseño y prueba de iniciativas de mejora de procesos.

2 Descripción de las herramientas

La suite de herramientas que se propone está integrada por tres herramientas que trabajan de manera colaborativa: SysDyn, que se encarga de ofrecer el entorno de diseño y pruebas funcionales de prototipos de procesos basados en simulación, y otras dos herramientas para dar soporte al despliegue de una estrategia de gamificación (OpenBadgesUCA) y al análisis inicial de los resultados de la misma (GamAnalyze).

A continuación se describen las características principales de cada de una de ellas y sus interrelaciones.

SysDyn. SysDyn tiene como objetivo principal permitir al usuario la creación de prototipos funcionales de procesos basados en simulación sobre dispositivos táctiles como tablets. Es una aplicación móvil para sistemas Android implementada de manera híbrida, es decir, combinando código nativo Android (Java), con HTML5, JavasScript y CSS3.

De su funcionalidad, cabe destacar los siguientes aspectos:

- Gestión de los usuarios: registro, sistema de login, cambio de contraseña, cierre de sesión del usuario, recomendación personalizada en Twitter.
- Gestión de modelos: Creación de modelos mediante Drag&Drop, edición y validación de las ecuaciones de los modelos, simulación de modelos siguiendo el método de Euler, representación gráfica de los resultados obtenidos en la simulación, funcionalidades básicas de modelos, como aumento y disminución de zoom, eliminación de componentes, funciones deshacer y rehacer, guardar y cargar modelos desde la memoria del dispositivo.

Además, al objeto de dar soporte a la gamificación del uso de la herramienta, SysDyn ofrece la siguiente funcionalidad específica: a) asignación de medallas en función de diferentes logros dentro del uso de la aplicación y b) envío de medallas a la cuenta de Mozilla OpenBadges del usuario.

OpenBadgesUCA. OpenBadgesUCA tiene como objetivo comunicar los logros reconocidos en aplicaciones gamificadas, como SysDyn, bajo la forma de medallas a la

cuenta de la que el usuario disponga en la plataforma Mozilla OpenBadges [3]. De esta manera, se consigue que los logros conseguidos en SysDyn tengan una visibilidad mayor y puedan ser compartidos en redes sociales profesionales como LinkedIn. OpenBadgesUCA se ha implementado como una web con un script PHP que mediante una URL recibe unos parámetros encriptados correspondientes a diferentes datos enviados por SysDyn, Su cometido es desencriptar esos datos y permitir al usuario la subida de las medallas a su cuenta de Mozilla OpenBadges. OpenBadgesUCA emplea HTML5, JavaScript, CSS3 y PHP. Su funcionalidad principal es la siguiente:

- Gestión de medallas: comprobar la validez de las medallas que el usuario desea subir a su cuenta de Mozilla OpenBadges y permitir al usuario subir las medallas a su cuenta.
- Gestión de la información: registrar en la base de datos información relativa al uso por parte de un usuario de la plataforma Mozilla OpenBadges y mostrar información sobre la propia aplicación.

GamAnalyze. GamAnalyze tiene como objetivo analizar los datos provenientes de las aplicaciones registradas en OpenBadgesUCA a fin de proporcionar información sobre los resultados de las estrategias de gamificación desplegadas en dichas aplicaciones. GamAnalyze es una aplicación web enfocada dentro del término Web 2.0. Esta aplicación se ha construido empleando yii Framework haciendo uso de todas sus características. La funcionalidad principal de GamAnalyze es la siguiente:

- Ayuda al análisis de las estrategias de gamificación: consultar los datos registrados por OpenBadgesUCA, realizar diferentes cálculos sobre los datos, mostrar los resultados gráficamente a fin de permitir su análisis por parte del usuario, presentar gráficas interactivas, exportación de gráficas en diferentes formatos, mostrar los datos de manera tabular y permitir diferentes criterios de ordenación y selección en las tablas de resultados. También permite la creación de medallas directamente en la aplicación, generando el fichero pertinente para Mozilla OpenBadges.
- Gestión de los usuarios: control de acceso basado en roles diferenciando entre administradores y usuarios sin privilegios, creación, actualización y borrado de usuarios para la propia plataforma de análisis, permitir el cambio de contraseña dentro de la aplicación, envío de correos para contactar con el desarrollador y de contraseña olvidada a los usuarios, integración de las diferentes redes sociales más conocidas dentro de la aplicación.

La figura 1 muestra las interrelaciones existentes entre las tres herramientas. Según se ha descrito, SysDyn colabora con OpenBadgesUCA en el reconocimiento de los logros obtenidos en su uso. OpenBadgesUCA se encarga de subir estos logros bajo la forma de medallas en Mozilla OpenBadges de manera que el usuario pueda compartirlas en sus redes sociales, si así lo desea, y también registra la información en una base de datos que utiliza GamAnalyze para permitir monitorizar y analizar los valores de indicadores principales de la evaluación de la estrategia de gamificación.

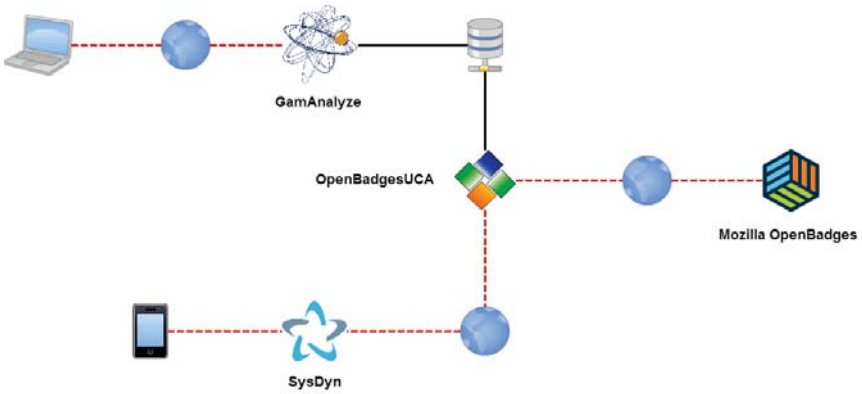


Fig. 1. Interrelaciones entre las herramientas colaborativas

3 Conclusiones y trabajo futuro

Las herramientas presentadas constituyen la primera versión de una suite de herramientas colaborativas que permita el despliegue y el análisis de los resultados de estrategias de gamificación enfocadas a motivar y a promover el compromiso e implicación de sus usuarios. En nuestro caso, se ha comenzado desplegando una estrategia, aún sencilla, de gamificación sobre una aplicación enfocada al diseño y pruebas de iniciativas de mejora de procesos que utiliza el prototipado funcional de procesos. Nuestras próximas tareas están enfocadas a la realización de experimentos con usuarios reales que permita evaluar la validez de la propuesta y su mejora.

Agradecimientos.

Este trabajo se ha llevado a cabo en el marco de los proyectos del Ministerio de Economía e Innovación (TIN-20057-C03-03 y TIN2013-46928-C3-2-R) y el proyecto TIC-195 de la Consejería de Economía, Innovación, Ciencia y Empleo de la Junta de Andalucía.

Referencias

1. Deterding S, Dixon D, Khaled R, and Lennart Nacke (2011). From game design elements to gamefulness: Defining "gamification". Proceedings of the 15th International Academic MindTrek Conference. pp. 9–15.
2. Grasl, Oliver. Business Prototyping (2014) http://www.business-prototyping.com/ [último acceso: mayo de 2014]
3. Mozilla. Get recognition for skills you learn anywhere (2014) http://openbadges.org/ [último acceso: mayo de 2014]

Entorno para la Evaluación y Certificación de la Calidad del Producto Software

Moisés Rodríguez[1], Mario Piattini[2]

[1] Alarcos Quality Center, Universidad de Castilla-La Mancha, Ciudad Real, España.
moises.rodriguez@alarcosqualitycenter.com
[2] Instituto de Tecnologías y Sistemas de la Información, Universidad de Castilla-La Mancha, Ciudad Real, España.
mario.piattini@uclm.es

Resumen. La calidad del software está adquiriendo durante los últimos años una gran importancia, principalmente debido a que el software está presente en prácticamente todo lo que nos rodea. Para poder controlar dicha calidad se hace necesario llevar a cabo evaluaciones del software, que inicialmente se centraron en los procesos de desarrollo y que durante los últimos años se están enfocando más en la calidad del propio producto software. Para poder evaluar la calidad de un producto software se requiere de varios elementos entre los que se pueden destacar: un modelo de calidad, las métricas que se van a utilizar, el proceso de evaluación y las herramientas de soporte que automaticen todo lo posible el proceso. El presente artículo expone un entorno integrado que permite llevar a cabo la evaluación de la calidad del producto software, alineando todos los elementos anteriores a un estándar internacional como es la nueva familia de normas ISO/IEC 25000. Dicho entorno se ha validado además mediante su aplicación en un laboratorio de evaluación acreditado y la realización de varios proyectos piloto con una entidad de certificación, cuyo resultado ha sido la obtención de los primeros productos certificados por AENOR en mantenibilidad software a nivel internacional.

Palabras clave: modelo de calidad, métricas de calidad, ISO/IEC 25000, mantenibilidad, laboratorio de evaluación, certificación del producto software.

1 Introducción

Hoy en día el software está en prácticamente todo lo que nos rodea, no solo en los ordenadores, móviles o tablets, sino en los electrodomésticos, en los vehículos, en el sector de la medicina, en la banca y seguros, etc. Este aumento en la demanda de productos software ha dado lugar a un crecimiento de las empresas y departamentos encargados de su desarrollo, lo que se conocen como "factorías de software". Por otro lado, la falta de personal especializado para ciertas tareas del desarrollo software, así como la búsqueda de la reducción de costes han dado lugar a lo que se conoce como "outsorcing" del desarrollo software, de manera que las empresas externalizan todo o parte de las actividades de desarrollo software a otros departamentos o empresas. Sin

embargo, cuando se externalizan actividades de desarrollo software, también aumentan los riesgos y la falta de control sobre la calidad del software que la empresa contratada entrega, surgiendo la necesidad de evaluar y asegurar la calidad del software de dichas empresas desarrollan.

Aunque desde hace bastantes años la evaluación de la calidad del software es un campo de gran actividad tanto investigadora como en el sector industrial, la mayor parte del esfuerzo realizado se ha centrado en la calidad de los procesos, habiéndose desarrollado gran cantidad de modelos y estándares de referencia, evaluación y mejora de procesos software: ISO 90003, ISO 12207, ISO 15504, CMM, CMMI, IDEAL, SCAMPI, etc., en los que numerosas empresas de todo el mundo se han evaluado y/o certificado. Sin embargo, hay poca evidencia de que cumplir un modelo de procesos asegure la calidad del producto software desarrollado, y aunque la estandarización de los procesos garantiza la uniformidad en la salida de los mismos, podría llegar a darse el caso de que institucionalizara la creación de malos productos [1]. En este sentido, nosotros estamos de acuerdo con que las evaluaciones deberían basarse en evidencias directas del propio producto, y no solo en evidencias del proceso de desarrollo [2]. Por ello, es cada día mayor el número de organizaciones y empresas que se interesan, no solo por la calidad de los procesos que se siguen en el desarrollo de software, sino también por la calidad de los productos que desarrollan y/o adquieren, ya que una vez que el producto ha sido implantado en sus instalaciones se encuentran con graves problemas de calidad y complicaciones a la hora de corregirlo, adaptarlo o evolucionarlo.

En los últimos años se han elaborado también trabajos de investigación, normas y estándares, con el objetivo de crear modelos, procesos y herramientas de evaluación de la calidad del propio producto software, entre los que se pueden destacar los presentados en [3, 4, 5, 6]. Precisamente para dar respuesta a estas necesidades nace la nueva familia de normas ISO/IEC 25000 conocida como SQuaRE (*Software Product Quality Requirements and Evaluation*), que tiene por objetivo la creación de un marco de trabajo común para evaluar la calidad del producto software, sustituyendo a las anteriores ISO/IEC 9126 e ISO/IEC 14598 y convirtiéndose así en el referente a seguir. La ISO/IEC 25000 se encuentra compuesta de varias partes o divisiones, entre las que podemos destacar la ISO/IEC 25040 [7] que define el proceso de evaluación de la calidad del producto software, y la ISO/IEC 25010 [8] que determina las características y subcaracterísticas de calidad que se pueden evaluar para un producto software (Fig. 1). Sin embargo, la división que define las métricas de calidad de la familia ISO/IEC 25000 todavía está pendiente de publicación, por lo que no existe un acuerdo respecto a los indicadores y umbrales que se deben considerar para poder determinar la calidad de un producto software de manera estandarizada.

Por otro lado, los modelos y normas anteriores relacionados con la evaluación del producto software, no tratan el proceso posterior de la certificación, que permita a las empresas superar una auditoría realizada por una entidad acreditada y obtener un certificado que refleje la calidad de su producto software. Por tanto la certificación de la calidad del producto software sigue siendo a día de hoy un área relativamente inmadura de la Ingeniería del Software, en la que todavía no existe un consenso definitivo.

Fig. 1. Modelo de calidad del producto software según la ISO/IEC 25010

Por este motivo y con el objetivo de conocer los trabajos existentes sobre certificación de la calidad del producto software, en 2012 se realizó una revisión sistemática [9] siguiendo la guía propuesta por Kitchenham en [10]. Como resultado se obtuvo un conjunto de 10 estudios primarios que cumplían con los requisitos de búsqueda: [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]. Entre las principales conclusiones obtenidas de esta revisión sistemática, se pueden destacar las siguientes:

- La mayoría de los estudios destacan la necesidad de que la certificación sea también extendida a las características del producto y no se centre sólo en el proceso.
- La mayoría de los estudios se basan en características de calidad extraídas de normas internacionales como la ISO/IEC 9126, utilizando métodos de evaluación también estándares como los de la ISO/IEC 14598. Sin embargo, ninguna de las propuestas ha adoptado el nuevo modelo y proceso propuestos por la familia ISO/IEC 25000.
- Aunque ya existen varias propuestas para certificar el producto software, la mayoría de ellas carece o tiene un número reducido de aplicaciones reales en la industria del software, lo que se considera vital a la hora de poder extender su aceptación a nivel internacional y poder contrastar los niveles obtenidos por diferentes productos software.
- La mayoría de los estudios utiliza indistintamente los conceptos de evaluación y certificación, lo que consideramos un error. Es necesario diferenciar entre el proceso de evaluación, realizado frente a un modelo de calidad, y el posterior proceso de certificación realizado por un organismo acreditado e independiente que asegure la validez del certificado emitido.
- Un denominador común de los estudios que presentan casos de aplicación práctica es la importancia que atribuyen a disponer de una herramienta que automatice las actividades. La mayoría de estos estudios carecen de dicha herramienta y se plantean como trabajo futuro abordar su construcción para que les permita medir, procesar los datos y analizar los resultados de una manera más rápida y sencilla.

Por todo lo anterior, durante los últimos años hemos trabajado en la construcción de un entorno para realizar evaluaciones de la calidad del producto, adoptando la familia de normas ISO/IEC 25000, que permita además a las empresas obtener una certificación del nivel de calidad alcanzado. Por ello, el objetivo del presente artículo es presentar las características principales de este entorno para la evaluación y certificación del producto software y demostrar su aplicación mediante un laboratorio acreditado, teniendo en cuenta que por cuestiones de espacio, hemos tenido que omitir ciertos detallalles del modelo, métricas y herramientas concretas utilizadas. Para ello, el resto del artículo se estructura de la siguiente manera: en el apartado 2 se presenta el modelo de calidad elaborado. En el apartado 3 se resumen las características del proceso de evaluación. Posteriormente en el apartado 4 se describe el entorno automatizado para la evaluación del producto. En el apartado 5 se resumen las principales características del laboratorio acreditado AQC Lab y su funcionamiento junto con AENOR para la evaluación y certificación del producto. Y finalmente, el apartado 6 presenta las conclusiones obtenidas con este trabajo y las líneas futuras de investigación.

2 Modelo de Calidad

El entorno de Evaluación descrito en el presente artículo se encuentra compuesto principalmente de tres elementos:

1. El **Modelo de Calidad**, que describe las características, subcaracterísticas, propiedades de calidad y métricas que se utilizan para determinar el valor de calidad de un producto software. En este elemento se centra el presente apartado.
2. El **Proceso de Evaluación**, que describe el conjunto de actividades y tareas que se realizan para llevar a cabo un proceso de evaluación del producto software.
3. El **Entorno Tecnológico**, formado por el conjunto de herramientas software que permiten automatizar la medición, aplicación de criterios de evaluación y visualización de los resultados.

De entre las características de calidad propuestas por la ISO/IEC 25010, inicialmente se decidió centrar el modelo de calidad en la característica de la mantenibilidad, entendida como el grado de efectividad y eficiencia con el que un producto puede ser modificado, debido principalmente a las siguientes razones:

- El mantenimiento supone una de las fases del ciclo de vida de desarrollo más costosa, sino la más, llegando a alcanzar el 60%.
- La mantenibilidad es una de las características más demandadas hoy en día por los clientes de software, que piden que el producto software que se les desarrolle pueda ser después mantenido por ellos mismos o incluso por un tercero.
- Las tareas de mantenimiento sobre productos con poca mantenibilidad tienen más probabilidad de introducir nuevos errores en el producto.
- Del estudio realizado en [9], se deduce que la mantenibilidad, junto con la funcionalidad, la usabilidad y la fiabilidad, es una de las características que más interesan desde el punto de vista de la certificación del producto.

El resto del presente artículo se centrará en esta característica, si bien el modelo de calidad ya se ha ampliado para la característica de funcionalidad y está en fase de completarse también para la usabilidad. Todo ello será publicado en la comunidad científica una vez se validen y se acrediten formalmente, igual que se ha hecho para la característica de la mantenibilidad.

2.1 Subcaracterísticas y propiedades de la mantenibilidad

El modelo de calidad definido para la mantenibilidad parte exactamente de las cinco subcaracterísticas de calidad definidas en la ISO/IEC 25010, que son:

- **Analizabilidad**. Se define como la facilidad para identificar las partes de un sistema que se deben modificar debido a deficiencias o fallos, o la capacidad de evaluar el impacto que puede provocar un cambio en el sistema.
- **Modularidad**. Se define como el grado, en el que un sistema se encuentra dividido en módulos de forma que el impacto que causa una modificación en un módulo sea mínimo para el resto.
- **Capacidad de ser Modificado**. Se define como el grado en el que se pueden realizar cambios en un producto software de forma efectiva y eficiente, sin introducir defectos ni degradar su rendimiento.
- **Capacidad de ser Reutilizado**. Se define como el grado en que un activo (módulo, paquete, clase, etc.) puede ser usado en más de un sistema o en la construcción de otros activos.
- **Capacidad de ser Probado**. Se define como la facilidad para establecer criterios de prueba para un sistema y realizar las pruebas que permitan comprobar que se cumplen esos criterios.

Sin embargo, la familia de normas ISO/IEC 25000 todavía no ha definido el conjunto de métricas e indicadores que afectan a cada una de estas subcaracterísticas, los umbrales para las mismas, ni las funciones de medición a aplicar para poder calcular el valor de calidad de cada una de ellas. Por ello, para completar este modelo de calidad y hacerlo operativo, se han identificado un conjunto de propiedades de calidad, que obtienen su valor a partir de métricas del código fuente, y se ha establecido la relación que existe con las subcaracterísticas anteriormente indicadas. El objetivo al identificar estas propiedades de calidad y métricas no ha sido que fuera el conjunto mayor posible, sino que fuera un grupo completo y sin lugar a controversias, basado en los estudios e investigaciones previas y aceptados por la comunidad científica. Estas propiedades de calidad son:

- **Incumplimiento de Reglas**. Esta propiedad se basa en el incumplimiento de reglas de programación y optimización de código establecidas por los estándares de programación de cada lenguaje y que se obtienen por varias herramientas de comprobación de reglas. Para cada subcaracterística de calidad se comprueba un conjunto de reglas relacionadas.
- **Documentación del Código**. La documentación de código hace referencia a los comentarios que existen en el código, los cuales se utilizan para explicar la funcio-

nalidad del código. Los comentarios influyen en la Analizabilidad, ya que los comentarios bien expresados ayudan a entender qué es lo que hace el código. Por esta misma razón, los comentarios también influyen en la capacidad para reutilizar el software: conocer qué es lo que hace un módulo ayuda a reutilizarlo en otros sistemas.

- **Complejidad**. La propiedad Complejidad se relaciona con la dificultad para implementar, probar, entender, modificar o mantener un programa. Como esta definición muestra, la Complejidad está relacionada con las subcaracterísticas: Analizabilidad, Capacidad de ser Modificado y Capacidad de ser Probado del modelo de calidad. En aplicaciones con complejidad elevada, las tareas de mantenimiento requieren más esfuerzo y, por tanto, son más costosas.
- **Estructuración**. La estructuración de un sistema es la división que se hace del mismo en componentes más pequeños. La calidad en el diseño de un sistema está estrechamente relacionada con esta propiedad: la estructuración correcta de un sistema en cualquiera de sus niveles (subsistemas, paquetes, clases, etc.) facilita su desarrollo y mantenimiento. Una correcta estructuración del sistema: facilita su comprensión y la identificación de elementos que se deben modificar (Analizabilidad), minimiza el impacto de los cambios que haya que realizar en el sistema (Modularidad), facilita la reutilización de únicamente aquellos elementos que sean necesarios, pudiendo utilizar toda la funcionalidad que se espera de ellos y no más o menos (Capacidad de ser Reutilizado) y facilita la elaboración de pruebas permitiendo que éstas se centren en comprobar el funcionamiento correcto de porciones concretas del sistema (Capacidad de ser Probado).
- **Tamaño de Métodos**. El tamaño de un sistema afecta directamente a la mantenibilidad del mismo, ya que, intuitivamente, un sistema más grande requiere más esfuerzo de mantenimiento. Se considera más adecuado evaluar el tamaño de sus elementos a un nivel más bajo, como por ejemplo a nivel de métodos, que evaluar directamente el tamaño global del sistema, ya que una modificación normalmente no afecta a todo el sistema, sino a un conjunto de sus elementos.
- **Código Duplicado**. Esta propiedad hace referencia a fragmentos de código que se encuentran repetidos en diferentes lugares del sistema. El código duplicado hace más difícil la realización de modificaciones en una aplicación software, ya que solucionar un defecto o introducir una mejora en dicho código requiere realizar las modificaciones en todas las partes del sistema en las que aparezca dicho código duplicado. El Código Duplicado afecta a las subcaracterísticas Analizabilidad y Capacidad de Ser Modificado, ya que el código duplicado dificulta la identificación del código que se debe modificar y dificulta la realización de modificaciones que sean efectivas. También afecta a la Subcaracterística Capacidad de Ser Probado, ya que se deben definir varios casos de prueba diferentes (aunque similares en forma) para comprobar el correcto funcionamiento del código duplicado en sus distintas instancias.
- **Acoplamiento**. El acoplamiento indica el grado de interdependencia entre las unidades de software (módulos, funciones, clases, bibliotecas, etc.). De forma general, cuanto más bajo sea el acoplamiento en una aplicación software, mejor se considera su diseño, ya que el bajo acoplamiento permite mejorar la mantenibilidad (los

cambios en una unidad no afectan al resto de unidades si no hay acoplamiento) y aumentar la reutilización de las unidades de software. El acoplamiento también dificulta la realización de pruebas, siendo más fácil probar elementos que dependen de pocos elementos externos [21].

- **Balance Inestabilidad - Abstracción**. La inestabilidad indica la relación entre el acoplamiento de salida y el acoplamiento total de un paquete y da una idea de la resistencia al cambio del mismo. La abstracción indica la relación entre clases abstractas (e interfaces) y el número de clases de un paquete. Debe existir un balance entre ambas, de forma que los paquetes deben ser abstractos y estables o concretos e inestables.
- **Ciclos**. La propiedad Ciclos hace referencia a la existencia de ciclos de dependencia entre los paquetes del sistema. El Principio de Dependencias Acíclicas [22] indica que no deben existir ciclos en la estructura de dependencias de un sistema. La existencia de Ciclos en un sistema afecta negativamente a su mantenibilidad dificultando la realización de modificaciones, ya que un cambio en un paquete repercute a todos los paquetes en el ciclo de dependencias [21]. Además, dificulta la reutilización del código y la realización de pruebas [22], ya que si se quiere reutilizar un paquete o realizar pruebas sobre alguna de sus clases es necesario tener presentes todos los paquetes involucrados en el ciclo.
- **Cohesión**. El concepto de cohesión [23] indica el grado de relación que existe entre los elementos internos de un módulo. Una clase tiene cohesión baja cuando realiza varias funciones no relacionadas. Así, las funcionalidades que proporciona mediante sus métodos tienen poco en común y para realizar sus actividades utiliza conjuntos de datos que no están relacionados. La Analizabilidad, la Modularidad y, en general, la Mantenibilidad en sistemas cuyas clases tienen una cohesión baja se ven perjudicadas, ya que estos sistemas son más difíciles de comprender, proporcionan funcionalidades que no son necesarias y su mala modularidad hace que los cambios en los requisitos afecten a varios módulos.

Tabla 1. Relación entre Propiedades y Subcaracterísticas de Mantenibilidad

Propiedades / Subcaracterísticas	Analizabilidad	Modularidad	Capacidad de ser Modificado	Capacidad de ser Reutilizado	Capacidad de ser Probado
Incumplimiento de Reglas	X	X	X	X	X
Documentación del Código	X			X	
Complejidad	X		X		X
Balance Inestabilidad-Abstracción / Acoplamiento		X	X	X	X
Ciclos		X	X	X	X
Cohesión	X	X			
Estructuración de Paquetes	X	X		X	X
Estructuración de Clases	X	X			X
Tamaño de Métodos	X	X			
Código Duplicado	X		X		X

En la Tabla 1 se resumen las relaciones existentes entre las subcaracterísticas y las propiedades de calidad. La relación entre una propiedad y una subcaracterística refleja que la primera influye en gran medida en el valor de calidad de la segunda. Que en dicha tabla no se indique relación entre una propiedad y una subcaracterística concretas no significa que la primera no influya en la segunda, sino que dicha influencia es menos relevante que la de otras propiedades.

2.2 Utilización del Modelo de Calidad

El valor de Mantenibilidad de un producto se obtiene a partir de los valores de calidad de sus subcaracterísticas. Para obtener la evaluación de las subcaracterísticas es necesario obtener el valor de las propiedades de calidad (vistas en el apartado anterior), que a su vez se evaluarán a través de los valores obtenidos para las métricas de las que dependen. Estas métricas son la parte base de la evaluación y su cálculo se realiza directamente a partir del código fuente del producto software que se está evaluando. En la Fig. 2 se puede observar la jerarquía de evaluación que se sigue en el modelo de calidad definido para determinar el valor de una característica de calidad.

La evaluación se realiza siguiendo un proceso *bottom-up*, de manera que primero se calculan las métricas de bajo nivel utilizando herramientas que procesan el código fuente del producto software bajo evaluación. Como vemos a la izquierda de la Fig. 2, cada una de estas métricas tiene un rango propio: número de clases, número de líneas de código duplicado, valor de la complejidad ciclomática de cada método, etc. A partir de los valores obtenidos en las métricas, se escala en la jerarquía al siguiente nivel para poder obtener un valor de calidad que asignarle a la propiedad que aplique. En este caso, el valor de las propiedades se ha normalizado entre 0 y 100 utilizando una función por perfiles (igual que se hará para las subcaracterísticas y la característica de calidad), de manera que todas las propiedades tengan la misma escala, siendo 0 el valor más bajo de calidad que puede tener una propiedad y 100 el valor más alto.

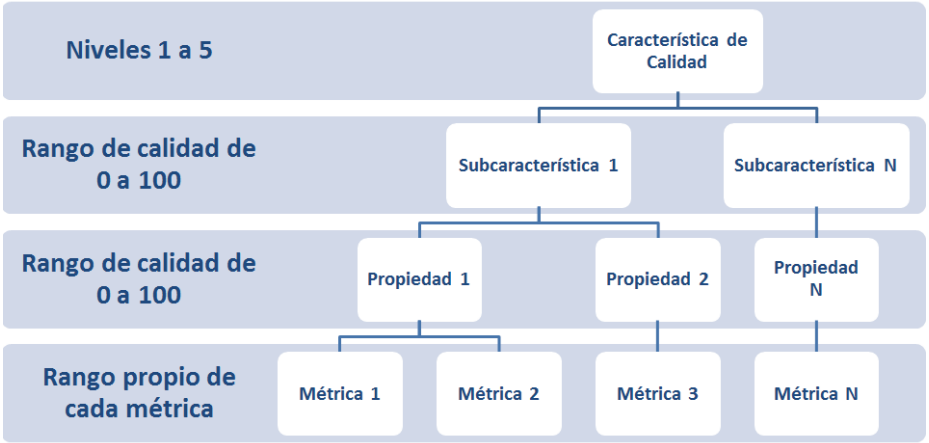


Fig. 2. Jerarquía y elementos que influyen en la evaluación del producto software

Una vez se dispone del valor de todas las propiedades, se pasa a calcular el valor de cada una de las subcaracterísticas de calidad. Cada una de estas subcaracterísticas estará influenciada por una o varias de las propiedades para las que previamente se ha calculado su valor. De nuevo, como se observa a la izquierda de la Fig. 2, el valor de las subcaracterísticas ha sido normalizado en un rango de 0 a 100, siendo igualmente el 0 el valor más bajo de calidad que puede obtener una subcaracterística y el 100 el valor más alto. Una vez se ha calculado el valor de todas la subcaracterísticas, estamos en disposición de calcular el nivel de calidad de la característica, en este caso mantenibilidad. Para ello, se aplica una función de evaluación sobre el conjunto de valores de 0 a 100 que presentan el grupo de subcaracterísticas y se obtiene un valor definitivo para la característica de calidad. En este caso, como se observa a la izquierda de la Fig. 2, el valor de calidad de las características se ha normalizado entre 1 y 5, en lugar de entre 0 y 100. La decisión de realizar esta clasificación por niveles se debe precisamente a cómo se encuentra hoy en día la cultura de la calidad del software, donde modelos anteriores para calidad de procesos ampliamente extendidos (CMMI, ISO/IEC 15504, etc.), siguen dicha clasificación por niveles de 1 a 5. La correspondencia de estos niveles de calidad con los valores normalizados de 0 a 100 siguen los rangos establecidos en la Tabla 2. Como se puede observar, un producto con una mantenibilidad en nivel 1 habrá obtenido un valor de calidad entre 0 y 25, por lo que se podrá considerar como deficiente. Una mantenibilidad en nivel 3, habrá obtenido un valor de calidad entre 50 y 75, por lo que se podrá considerar buena. Y una mantenibilidad en el nivel 5 habrá obtenido un valor de calidad entre 95 y 100, por lo que se podrá considerar como excelente.

Tabla 2. Niveles de Calidad para la Mantenibilidad del Producto Software

Nivel	Valor de Calidad	Descripción
1	0-25	Calidad Deficiente
2	25-50	Calidad Insuficiente
3	50-75	Calidad Buena
4	75-95	Calidad Muy Buena
5	95-100	Calidad Excelente

Aunque queda fuera del alcance del presente artículo, se debe indicar que la influencia que tienen las métricas y propiedades seleccionadas sobre la mantenibilidad ha sido validada. Como indica la norma ISO/IEC 25020 [24], uno de los métodos para demostrar la validez de una medición consiste en comprobar si existe correlación entre la característica de calidad y los valores medidos. Así, para realizar la validación se ha utilizado un método empírico basado en esta definición. En concreto, se ha realizado un caso de estudio en el que se han evaluado 30 proyectos reales y se ha comprobado que los valores de calidad proporcionados por el método están correlacionados con un indicador de esfuerzo de mantenimiento, como es el tiempo medio empleado en la resolución de defectos encontrados en fase de mantenimiento.

3 Proceso de Evaluación

Para la elaboración de este proceso de evaluación se tomaron como referencia las experiencias previas que se tenían en la construcción de metodologías de evaluación, como la expuesta en [25], pero principalmente la norma ISO/IEC 25040 [7], que describe el modelo de referencia para la evaluación del producto, formado por entradas, salidas, restricciones, recursos y actividades y sus tareas, estas dos últimas son:

1. Establecer los requisitos de la evaluación:
   (a) Establecer el propósito de la evaluación.
   (b) Obtener los requisitos de calidad del producto.
   (c) Identificar las partes del producto software que se van a evaluar.
   (d) Definir el rigor de la evaluación.
2. Especificar la evaluación:
   (a) Seleccionar las métricas de calidad (módulo de evaluación).
   (b) Definir los criterios de decisión para las métricas.
   (c) Establecer los criterios de decisión para la evaluación.
3. Diseñar la evaluación:
   (a) Planificar las actividades de la evaluación.
4. Ejecutar la evaluación:
   (a) Realizar las mediciones.
   (b) Aplicar los criterios de decisión a las medidas de calidad.
   (c) Aplicar los criterios de decisión a la evaluación.
5. Concluir la evaluación:
   (a) Revisar los resultados de la evaluación.
   (b) Crear el informe de evaluación.
   (c) Revisar la calidad de la evaluación.
   (d) Realizar la disposición o tratamiento de los datos

El proceso de evaluación construido adopta estas actividades y tareas, realizando una instanciación de las entradas, salidas, recursos y roles concretos que intervienen en nuestro proceso. En la Fig. 3 se expone un ejemplo para una de las actividades:

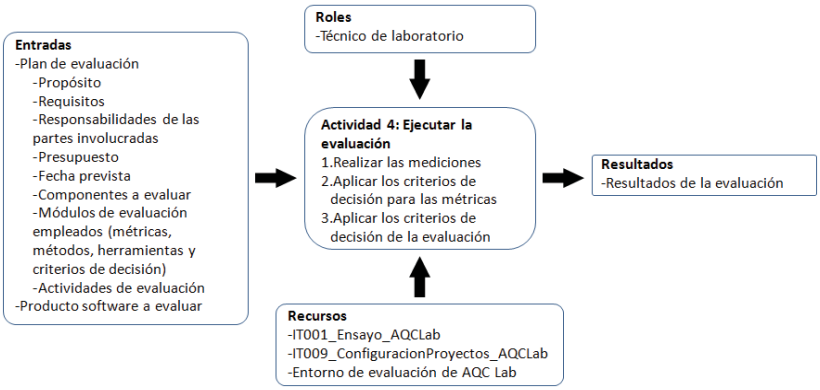


Fig. 3. Diagrama de la actividad “Ejecutar la Evaluación”

4 Entorno Tecnológico

El tercer elemento es el entorno tecnológico, para cuya construcción se partió de las experiencias previas que se tenían en la construcción de cuadros de mando relacionados con la calidad del producto [26]. El entorno construido permite automatizar hasta en un 90% las evaluaciones y está formado por tres niveles diferenciados:

1. **Herramientas de medición**. Constituyen el primer nivel y su misión es analizar el código fuente y generar archivos (normalmente XML) con los datos sobre métricas base y/o incumplimientos de reglas de programación para lenguajes como Java, .NET, PHP, C++, ObjetiveC, Android y Groovy. La ventaja de este nivel es que es fácilmente ampliable, añadiendo nuevas herramientas que permitan medir nuevos lenguajes de programación o calcular métricas para otras características de calidad.
2. **Sistema de evaluación**. Supone el nivel intermedio del entorno y su objetivo es analizar el conjunto de archivos generados por el nivel inferior y aplicar los criterios de evaluación del modelo de calidad, obteniendo como resultado los valores para las propiedades, subcaracterísticas y características de calidad.
3. **Entorno de visualización**. Representa el nivel superior del entorno y permite presentar de manera comprensible la información obtenida tras la evaluación del producto software. Además de mostrar los valores de calidad para propiedades, subcaracterísticas y características de calidad, este entorno permite la obtención de históricos para varias versiones de un producto, comparativas entre diferentes productos (Fig. 4) e incluso la generación de informes predefinidos.

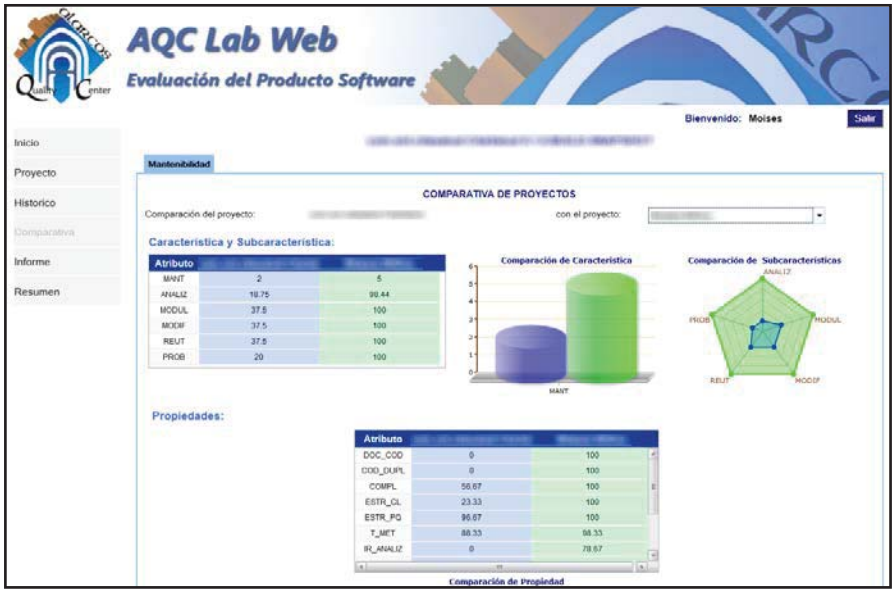


Fig. 4. Entorno de Visualización de los Resultados de Evaluación

5 Laboratorio de Evaluación

En paralelo a la última fase de construcción del entorno expuesto en los apartados anteriores y con el objetivo de obtener un reconocimiento a la validez de las evaluaciones realizadas mediante el mismo, se decidió elaborar toda la infraestructura de gestión necesaria para la creación de un laboratorio acreditado, siguiendo además para ello prácticas del desarrollo ágil como se expone en [27]. El resultado fue que en 2012 AQC Lab conseguía la acreditación de ENAC (Entidad Nacional de Acreditación) en la norma ISO/IEC 17025, como el primer laboratorio para la evaluación de la calidad de aplicaciones software bajo la familia de normas ISO/IEC 25000 (Fig. 5). La acreditación de acuerdo a la Norma ISO/IEC 17025 confirma la competencia técnica del laboratorio y garantiza la fiabilidad en los resultados de los ensayos realizados. Para ello, ENAC realiza una minuciosa evaluación de todos los factores que contribuyen a que los laboratorios obtengan resultados fiables:

- Son organizaciones que cuentan con personal cualificado y con experiencia.
- Disponen del equipamiento y de las infraestructuras necesarias.
- Aplican métodos y procedimientos de trabajo validados y apropiados.
- Emplean técnicas de evaluación de la calidad de los resultados.
- Emitiendo informes de evaluación claros y precisos.
- Cuentan con un sistema de aseguramiento de la calidad para gestionar su actividad.

Una vez alcanzada la acreditación del laboratorio, se estableció un proceso de trabajo con AENOR (Asociación Española de Normalización y Certificación) que permitiera que los productos software una vez se hubieran evaluado y obtenido un nivel adecuado de calidad, pudieran también conseguir un certificado. Como resultado AENOR creó un procedimiento que, a partir del informe del laboratorio acreditado y tras una auditoría, permitía certificar la calidad del producto software bajo estudio. Finalmente, a lo largo de 2013 se realizó un proyecto piloto de evaluación y certificación de los primeros productos software. Dicho proyecto permitió a tres empresas españolas evaluar, mejorar y finalmente certificar la calidad de sus productos software en base a la ISO/IEC 25000, tal y como se presenta en [28]. Entre los beneficios obtenidos, dichas empresas destacaron haber reducido hasta en un 40% el esfuerzo en las tareas de mantenimiento y hasta en un 30% el tamaño de sus productos.

Fig. 5. Marca de acreditación ILAC-ENAC con número de laboratorio AQC Lab

6 Conclusiones y Trabajo Futuro

La calidad del producto software es una de las principales preocupaciones tanto para las empresas desarrolladoras, como para los organismos que los adquieren. El presente artículo ha presentado una solución, formada por un modelo, un proceso y un entorno tecnológico, que permiten realizar la evaluación del producto software y que han sido validados y posteriormente acreditados por ENAC, consiguiendo además que los productos software puedan obtener un certificado de calidad emitido por AENOR. La experiencia del proyecto piloto nos ha permitido por un lado conocer cómo se encuentra la calidad real del producto software que desarrollan las empresas de nuestro país, que inicialmente obtuvo unos niveles bajos, pero que con el esfuerzo de refactorización les permitió obtener el certificado. Y por otro lado, detectar la necesidad real expresada por estas empresas, de difundir la posibilidad e importancia de evaluar la calidad de los productos software.

Como trabajo futuro destacamos la ampliación del entorno para la evaluación del resto de características de la familia ISO/IEC 25000, así como la inclusión de nuevas herramientas de medición que permitan analizar productos software desarrollados en lenguajes como Cobol, Ruby o PL/SQL, puesto que aunque las métricas son por norma general independientes del lenguaje, las herramientas de medición del primer nivel comentado en el entorno de evaluación, sí que lo son.

Agradecimientos. Esta investigación forma parte del proyecto ECU: Evaluación y Certificación del Producto Software (Consejería de Empleo y Economía y FEDER, 1313CALT0056) y el proyecto GEODAS-BC (Ministerio de Economía y Competitividad y FEDER, TIN2012-37493-C03-01).

Referencias

1. Kitchenham, B.,Pfleeger, S.L., *Software Quality: The Elusive Target.* IEEE Software, 1996. **20**(1): p. 12-21.
2. Maibaum, T.,Wassyng, A., *A Product-Focused Approach to Software Certification.* Computer, 2008. **41**(2): p. 91-93.
3. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., Macleod, G.J., Merrit, M.J., *Characteristics of Software Quality*. 1978: North-Holland.
4. ISO, *ISO/IEC 9126, Software Product Evaluation–Quality Characteristics and Guidelines for their Use*. 2001, International Organization for Standardization.
5. ISO, *ISO/IEC 14598-5:1998 - Information technology -- Software product evaluation -- Part 5: Process for evaluators.* 1998, International Organization for Standardization: Ginebra.
6. Heitlager, I., Kuipers, T., Visser, J., *A Practical Model for Measuring Maintainability*, in *Quality of Information and Communications Technology, 2007. QUATIC 2007.* 2007. p. 30-39.
7. ISO, *ISO/IEC 25040 Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Evaluation process*. 2011: Ginebra, Suiza.
8. ISO, *ISO/IEC 25010, Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. 2011: Ginebra, Suiza.

9. Rodríguez, M.,Piattini, M., *Systematic review of software product certification*, in *CISTI 2012: 7th Iberian Conference on Information Systems and Technologies*. 2012: Madrid. p. 631-636.
10. Kitchenham, B., *Guideline for performing Systematic Literature Reviews in Software Engineering. Version 2.3*. 2007, University of Keele (Software Engineering Group, School of Computer Science and Mathematics) and Durham (Department of Computer Science).
11. Heck, P., Klabbers, M., van Eekelen, M., *A software product certification model*. Software Quality Journal, 2009. **18**(1): p. 37-55.
12. Carvalho, F., Meira, S.R.L., Freitas, B., Eulino, J., *Embedded software component quality and certification.* Conference Proceedings of the EUROMICRO, 2009: p. 420-427.
13. Burger, E.,Reussner, R., *Performance certification of software components.* Electronic Notes in Theoretical Computer Science, 2011. **279**(2): p. 33-41.
14. Serebrenik, A., Mishra, A., Delissen, T., Klabbers, M., *Requirements certification for offshoring using LSPCM*. Proceedings - 7th International Conference on the Quality of Information and Communications Technology, QUATIC 2010, 2010: p. 177-182.
15. Yahaya, J.H., Deraman, A., Hamdan, A.R., *SCfM_PROD: A software product certification model*. 2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA, 2008.
16. Yahaya, J., Deraman, A., Hamdan, A.R., *Continuously ensuring quality through software product certification: A case study*. 2010 International Conference on Information Society, i-Society 2010, 2010: p. 183-188.
17. Hatcliff, J., Heimdahl, M., Lawford, M., Maibaum, T., Wassyng, A., Wurden, F., *A Software Certification Consortium and its Top 9 Hurdles.* Electronic Notes in Theoretical Computer Science, 2009. **238**(4): p. 11-17.
18. Morris, J., Lee, G., Parker, K., Bundell, G.A., Lam, C.P., *Software component certification.* Computer, 2001. **34**(9): p. 30-36.
19. Baggen, R., Correia, J.P., Schill, K., Visser, J., *Standardized code quality benchmarking for improving software maintainability.* Software Quality Journal, 2012. **20**(2): p. 287-307
20. Alvaro, A., De Almeida, E.S., Meira, S.L., *Towards a software component certification framework.* Proceedings - International Conference on Quality Software, 2007: p. 298-303.
21. Spinellis, D., *Code Quality: The Open Source Perspective*. 1 ed. Effective Software Development Series. 2006: Addison-Wesley Professional.
22. Martin, R.C., *Granularity*, in *C++ Report*. 1996. p. 57–62.
23. DeMarco, T., *Structured Analysis and System Specification.* 1979: Yourdon Press Computing Series.
24. ISO, *ISO/IEC 25020 Software and system engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Measurement reference model and guide*, in *International Organization for Standardization*. 2005: Ginebra, Suiza.
25. Rodríguez, M., Genero, M., Torre, D., Blasco, B., Piattini, M., *A Methodology for Continuous Quality Assessment of Software Artefacts*, in *The 10th International Conference on Quality Software (QSIC 2010)*. 2010: Zhangjiajie (China). p. 254-261.
26. Rodríguez, M., Garzás, J., Piattini, M., *Software Product Quality: An Open Source Automated Measurement Environment*, in *11th International Conference on Quality Engineering in Software Technology (CONQUEST 08)*. 2008: Postdam (Alemania).
27. Verdugo, J., Rodríguez, M., Piattinni, M., *Using Agile Methods to Implement a Laboratory for Software Product Quality Evaluation*, in *15th International Conference on Agile Software Development (XP 2014)*. 2014: Roma (Italia).
28. Rodríguez, M., Fernández, C.M., Piattini, M., *ISO/IEC 25000 Calidad del Producto Software*. AENOR. Revista de la Normalización y la Certificación, 2013(288): p. 30-35.

Generación de Mutantes Válidos en el Lenguaje de Programación C++

Pedro Delgado-Pérez, Inmaculada Medina-Bulo, Juan José Domínguez-Jiménez

Grupo UCASE de Ingeniería del Software, Departamento de Ingeniería Informática,
Universidad de Cádiz, Cádiz, España
{pedro.delgado,inmaculada.medina,juanjose.dominguez}@uca.es

Resumen La prueba de mutaciones es una técnica basada en fallos que se ha desarrollado alrededor de un amplio rango de lenguajes de programación. Sin embargo, la construcción de un marco de trabajo de prueba de mutaciones no comercial para C++ ha sido pospuesto en favor de otros lenguajes, principalmente por la variedad de alternativas que ofrece C++. Este artículo presenta una solución factible y completa para la implementación de los operadores de mutación en C++, la cual se basa en la búsqueda de patrones en el árbol de sintaxis abstracta (AST) que el compilador *Clang* genera a partir del código fuente. Estos patrones se construyen según las reglas que determinan los distintos operadores de mutación, permitiendo localizar los puntos del código en los que es posible introducir una mutación. Asimismo, en el artículo se abordan distintas situaciones que han de ser consideradas para la validez de los mutantes creados. Este proceso se ilustra a través de un operador de mutación a nivel de clase, si bien este enfoque sirve para crear operadores a cualquier nivel del lenguaje.

Palabras clave: Prueba de mutaciones, C++, árbol de sintaxis abstracta, operador de mutación

1. Introducción

La prueba de mutaciones es una técnica basada en errores que busca determinar la efectividad de los conjuntos de casos de prueba para detectar fallos presentes en el código [13]. Esta técnica consiste en introducir simples cambios sintácticos en el programa original mediante los *operadores de mutación*. Cada uno de estos cambios genera un nuevo programa al que se le conoce como *mutante*. Un buen conjunto de casos de prueba debe poder detectar todos los cambios que pueda presentar un programa afectando a su funcionamiento. Esa detección significa que el mutante produce un resultado distinto a la ejecución del programa original. En ese caso decimos que el mutante está *muerto* y, en el caso contrario, que el mutante sigue *vivo*.

En la actualidad existe una gran variedad de herramientas que emplean esta técnica con éxito para diferentes lenguajes [11] como *MuJava* para Java, *Proteum/IM 2.0* para C o *SQLMutation* para SQL. No obstante, el desarrollo con

respecto a C++ no está avanzado, a pesar de ser uno de los lenguajes orientados a objetos más conocidos y empleados, pudiendo solo encontrar algunas herramientas comerciales que realizan algunas mutaciones simples. Este hecho es debido principalmente a la dificultad de determinar las localizaciones en el código donde se puede inyectar una mutación por la complejidad del lenguaje. En [7], se presentó un conjunto de operadores de mutación a nivel de clase para C++ que supone un primer paso necesario antes de la implementación de un marco de trabajo de mutaciones.

Este artículo tiene como finalidad presentar un enfoque para la implementación de los operadores de mutación, que se ha empleado ya con éxito para sustentar la creación de una herramienta de prueba de mutaciones, de modo que el código fuente pueda ser analizado respecto al conjunto de operadores definido. Esta técnica de implementación está basada en el recorrido del árbol de sintaxis abstracta (AST), que es una forma intermedia que el compilador genera para representar el código de una manera estructurada. El AST que crea el compilador *Clang* es el empleado, buscando mediante patrones los nodos del mismo que cumplen los criterios que se establecen en los operadores de mutación. Este compilador es muy conveniente para esta tarea ya que es posible reutilizar las bibliotecas que ofrece para realizar las transformaciones del código necesarias.

El documento se estructura de la siguiente manera. En la Sección 2 se abordará la problemática de la prueba de mutaciones respecto a C++. La Sección 3 explica en detalle el método de implementación de los operadores de mutación a través del AST y qué ventajas nos ofrece el compilador seleccionado para la inyección de mutaciones. Además, se exploran distintas situaciones que han de ser consideradas para que los mutantes generados sean válidos. Por su parte, en la Sección 4 se mostrará un ejemplo concreto del uso de la técnica propuesta mediante el operador *CDD* [7], el cual es un operador a nivel de clase, aunque este modo de implementación es general para todas las construcciones del lenguaje. Finalmente, se expondrán las conclusiones y se sentarán las bases para el trabajo en el futuro.

2. Prueba de Mutaciones en C++

La prueba de mutaciones es una técnica que tiene como primer objetivo el establecer la utilidad del conjunto de casos de prueba definido para un programa en la detección de fallos en el código. En segundo lugar, pretende servir como eje para el proceso de mejora del conjunto de esos casos de prueba a fin de que descubran las desviaciones introducidas por los distintos operadores de mutación. Estos operadores se obtienen del análisis de los errores más habituales que los programadores suelen cometer y están específicamente diseñados de acuerdo a las características de cada lenguaje de programación. Por ejemplo, un fragmento de código como $x + 1$ puede ser mutado a $x - 1$ por un operador que cambie el operador aritmético. Este cambio producirá un nuevo programa (mutante) que solo difiere del original en esta modificación.

Para la evaluación de la efectividad del conjunto de casos de prueba se emplea la puntuación de mutación, que calcula el número de mutantes cuyo fallo ha sido detectado por algún caso de prueba al producir una salida distinta a la del programa original (mutante muerto), entre el número de mutantes total. A este total es necesario restarle el número de mutantes equivalentes, es decir, aquellos mutantes que, a pesar del cambio introducido, siguen preservando el mismo comportamiento que el programa original y no es posible encontrar un caso de prueba para su detección. La inclusión de nuevos casos de prueba que maten mutantes cuyo cambio seguía sin ser revelado (mutantes vivos), vendrá a incrementar la puntuación de mutación.

En sus inicios, la prueba de mutaciones se desarrolló en torno a lenguajes de programación estructurada, por lo que la mayoría de estudios y herramientas que implementan esta técnica están centrados en lenguajes como Ada, C o Fortran. Sin embargo, a partir de los años 90, la aparición y aumento del uso de lenguajes de orientación a objetos (OO) creó nuevas líneas de investigación respecto a estos lenguajes. Esto es debido a que los operadores tradicionales creados anteriormente no eran suficientes para cubrir los potenciales fallos que podían causar las nuevas estructuras involucradas en este paradigma. En este sentido, el lenguaje Java ha sido el que ha acumulado una mayor atención, así como el lenguaje C#, pudiendo encontrar multitud de artículos de estudios sobre los operadores [10,12], y herramientas para estos lenguajes [11].

Sin embargo, se puede afirmar que el estudio de esta técnica para su uso sobre el lenguaje C++ no ha sido afrontado en la literatura al mismo nivel que estos dos lenguajes OO. Por una parte, solo se pueden encontrar herramientas comerciales (ver Cuadro 1) que ofrecen muy poca información y, además, aplican la técnica de una manera selectiva en general, implementando solo algunas mutaciones simples. Respecto al conjunto de operadores de mutación, en un trabajo previo se definió un catálogo de operadores al nivel de clase para este lenguaje [7], ya que tampoco existía un conjunto de operadores definidos para C++ hasta el momento y solo en [9] se pudo encontrar una enumeración de varios errores comunes que pueden cometer los programadores en sistemas OO (pero sin detallar los operadores concretos para el lenguaje). Un estudio de la prueba de mutaciones en torno a C++ con información más detallada se llevó a cabo en [6], donde se comentan posibles motivos por los cuales este lenguaje no ha sido abordado en profundidad hasta el momento.

Una vez definido el conjunto de operadores, se hace necesario buscar la forma de automatizarlos para que sea posible analizar el código fuente respecto a estos operadores y poder realizar las mutaciones pertinentes, hecho que constituye la base de una herramienta de prueba de mutaciones. En cuanto al sistema de inyección de mutaciones para herramientas de lenguajes similares, distintas técnicas han sido utilizadas como la inserción de fallos en el *bytecode* o el análisis de código mediante expresiones regulares. El enfoque que se presenta en este artículo se aproxima más al modo empleado en *ILMutator* [8], en el cual se determinan las localizaciones en las que introducir la mutación en una representación intermedia del código.

Tabla 1. Herramientas comerciales dedicadas a C++

Nombre	Ref.	Año	Descripción general
Insure++	[3]	1998	Enfoque distinto a la prueba de mutaciones clásica: solo se generan mutantes funcionalmente equivalentes, los cuales se espera que sigan vivos tras su ejecución.
Plextest	[5]	2005	Trata de eludir la creación de mutantes equivalentes realizando una mutación altamente selectiva a través de la mutación de borrado.
Certitude	[1]	2006	Aunque puede adaptarse a nuevos lenguajes, actualmente se centra en la microelectrónica para simular el funcionamiento que tendrá un diseño abstracto de un circuito digital.

3. Transformaciones de Código mediante el Árbol de Sintaxis Abstracta

3.1. Utilización del Árbol de Sintaxis Abstracta del compilador Clang

La técnica empleada para la inserción de las desviaciones en el código es un punto determinante en la prueba de mutaciones. Para ello es necesario recurrir a un método para el análisis del código. En el caso de C++, un análisis directo sobre el código a alto nivel mediante la búsqueda de expresiones regulares puede ser insuficiente, por ejemplo, cuando se requiere evaluar el contexto de los elementos involucrados. De este modo, el análisis de la estructura de datos empleada internamente por un compilador para este lenguaje parece una solución más adecuada. En este sentido, el árbol de sintaxis abstracta o AST es la representación más ampliamente usada en torno a lenguajes de propósito general. Este árbol es la forma intermedia del proceso de compilación, resultante del analizador sintáctico, y contiene la estructura de cada expresión representada como ramas del árbol con la información esencial del código. De esta manera, el AST facilita la búsqueda de elementos concretos y, además, nos proporciona una mayor consistencia en el proceso de análisis, ya que los distintos elementos están representados de forma uniforme, permitiéndonos obviar detalles más específicos del código.

LLVM [4] es un proyecto que tiene como finalidad la creación de nuevos compiladores optimizados para cualquier lenguaje de programación. Para ello, *LLVM* suministra la infraestructura necesaria para el desarrollo de compiladores, actuando como backend al tomar el código intermedio que los distintos frontends para diversos lenguajes generan. Se trata de un proyecto de código abierto que engloba un número de subproyectos para la creación de nuevos frontends que trabajen sobre *LLVM*, muchos de los cuales son empleados en otros proyectos, tanto comerciales como de código abierto, así como en el ámbito académico.

Clang [2], por su parte, es uno de estos frontends desarrollados para el proyecto *LLVM* y que está dedicado a la familia de lenguajes C: C, C++, Objective-C

and Objective C++. Este frontend fue construido como parte nativa del sistema *LLVM* (inicialmente *LLVM* estuvo enfocado a C/C++) y, a día de hoy, se distribuye en sus nuevas versiones. La unión de ambos, *LLVM* y *Clang*, constituye una cadena de herramientas completa y funcional que puede ser usada para crear nuevas herramientas de análisis de código, siendo este el objetivo de nuestro estudio. Este diseño es una de las mayores ventajas del uso de este compilador, ya que nos permite embeber fácilmente el compilador en las nuevas aplicaciones creadas, lo que no es tan fácil en otros compiladores, como por ejemplo *GCC*.

Teniendo en cuenta que la prueba de mutaciones necesita analizar y cambiar el código respecto a los operadores de mutación al ser una técnica de caja blanca, el uso de estas bibliotecas es realmente útil como base para la construcción de una herramienta de prueba de mutaciones. Este análisis se consigue a través del AST, realizando búsquedas adecuadas a los requerimientos de los operadores de mutación (la forma de realizar esta búsqueda es abordada con más detalle en la Sección 3.2). Algunas de las características de *Clang* que parecen más apropiadas para el propósito de la técnica son:

- El AST que se genera tiene forma de documento XML, que es fácilmente comprensible sin necesidad de tener un conocimiento avanzado de cómo funciona el compilador. Este hecho nos permite comprender los diferentes casos en los que el operador de mutación debe o no actuar, o cómo hacer la búsqueda concreta de un nodo dentro del árbol.
- *Clang* guarda un registro de información de cada token en el código, lo cual nos permite localizar el rango de código que se corresponde con el nodo o nodos que están involucrados en la mutación.
- Se mantiene el mismo código en todo momento, es decir, que al convertir el árbol a forma de código se vuelve a obtener el código original. Esto es algo clave en los mutantes ya que estos deben ser iguales que el programa original excepto por el cambio que se ha introducido.

3.2. Implementación de los Operadores de Mutación

El método de implementación de los operadores propuesto reutiliza el AST que el compilador *Clang* [2] genera internamente como representación intermedia del código. Este árbol representa los distintos elementos del lenguaje de forma estructurada y simplificada. De esta manera, el árbol puede ser recorrido en busca de los nodos que cumplen las características prescritas en los operadores de mutación. Cada elemento del lenguaje es representado con un tipo de nodo en el AST, teniendo cada nodo una clase específica para manejar sus atributos y las relaciones existentes con otros nodos.

En este sentido, *Clang* proporciona sus bibliotecas para poder analizar convenientemente las diferentes construcciones del lenguaje. Estas bibliotecas incluyen un lenguaje de dominio específico (DSL) para crear patrones de búsqueda, conocidos como *matchers*, que están basados en los miembros de las clases de los distintos tipos de nodos. De esta manera se puede generar un patrón particular que se ajuste a las reglas de cada operador. Como ejemplo a alto nivel, si un

operador necesitase recuperar los nodos pertenecientes a los distintos métodos que están declarados en cada clase de un programa dado, se podría ejecutar el siguiente patrón:

Para cada **clase x** *perteneciente al* **programa L** *hacer*
    Para cada **método y** *perteneciente a la* **clase x** *hacer*
        recuperar **método y**
    Fin para cada
Fin para cada

Este DSL incluye unos *matchers* ya definidos, pudiendo combinar distintos patrones simples para crear predicados más complejos que representen a cada operador de mutación. También es posible definir nuevos patrones para el uso de los miembros de las clases en las bibliotecas que no están contemplados en la lista de *matchers* que ofrece *Clang*. Un ejemplo de composición de diferentes *matchers* para crear un operador de mutación se muestra en la Sección 4.

Este enfoque nos permite acometer dos de las principales fases en la prueba de mutaciones, el análisis y la generación de los mutantes, las cuales se describen a continuación:

- *Análisis*: El AST es recorrido en busca de localizaciones oportunas respecto a los patrones creados previamente. Todos los patrones pueden ser buscados de una vez en un único recorrido del árbol. Una vez encontrados los nodos potenciales, es posible estudiar algunas otras condiciones, recolectar información útil para la mutación,... Este post-procesado es necesario en ciertas situaciones ya que este lenguaje de dominio específico puede ser insuficiente cuando se realizan comprobaciones complejas, teniendo en cuenta que no existen *matchers* para todos los métodos de la API (aunque también es posible diseñar nuevos invocando a los métodos de estas bibliotecas).
- *Generación*: Los nodos detectados en el análisis son manipulados adecuadamente para insertar la mutación oportuna. Una vez realizada la modificación del operador, se puede regresar a la forma original de código fuente gracias a las bibliotecas del compilador. Este código mutado puede ser guardado en ficheros que pasarían a ser los mutantes.

Como se ha comentado en la Sección 3.1, *LLVM* está diseñado para permitir la inclusión de nuevos frontends para cualquier lenguaje. De esta manera, el modo de implementación de operadores descrito puede ser aplicado a otros lenguajes, siempre que el compilador relativo al lenguaje estudiado ofrezca los mismos recursos para el análisis de código.

3.3. Mutaciones Válidas

Los operadores de mutación a nivel de clase para C++ definidos en [7] presentan ciertas diferencias con respecto a la definición tomada en otros artículos y herramientas, debido a las características particulares de cada lenguaje. Asimismo, a

la hora de implementar los operadores, es necesario tener en cuenta las peculiaridades propias del lenguaje para que los mutantes que se creen sean los esperados y no se generen un mayor o un menor número de ellos. Además, también se hace necesario manejar otros aspectos que pueden llevar a una mutación no deseada.

En este sentido, se detectaron diversas situaciones que deben ser controladas a tal efecto a través del uso de esta técnica con diversos programas escritos en C++, tanto programas para la explicación de diversas características del lenguaje como aplicaciones de código abierto. Las consideraciones más relevantes son comentadas a continuación:

1. Mutantes duplicados: En algunos operadores es posible la creación de dos o más mutantes iguales. Esto se produce en los operadores que tienen que ver con la herencia en el conjunto de operadores de clase a causa de la herencia múltiple. Por ejemplo, el operador *IOD* o "borrado de método sobrescrito", que elimina métodos que ocultan a los de las clases de las que heredan, podría encontrar el mismo método en varias de las clases en su recorrido por las clases padres, por lo que se generaría el mismo mutante si este hecho no es tenido en cuenta.
   También se ha detectado otra situación que conlleva a la creación de mutantes duplicados: la utilización de plantillas. Una especialización de plantilla de una clase también se explicita en el AST como una clase más, así que un elemento encontrado en el AST puede ser parte de esa especialización concreta. Por consiguiente, si las especializaciones de plantilla no son excluidas del proceso, se produciría el mismo mutante por cada especialización contemplada en el código.
   La detección de estos mutantes puede llevarse a cabo en el proceso de generación de mutantes teniendo en cuenta estos supuestos en la implementación de los operadores.
2. Elementos implícitos: El AST generado a partir del código fuente puede contener elementos implícitos, es decir, construcciones del lenguaje que no están explícitamente escritas en el código. Como ejemplo, podemos citar ciertos constructores y destructores de clases en las que el programador no haya definido estos métodos, o invocaciones a constructores de las clases bases en las inicializaciones de un constructor. Así, es necesario excluir los elementos implícitos para que no sean:

   a) tomados como localizaciones de mutación.
   b) utilizados como elementos de reemplazo.
   c) tenidos en cuenta en la inyección de la mutación. Por ejemplo, en el operador *CID* o "borrado de la inicialización de una variable miembro", que elimina inicializaciones de la lista de inicialización, es importante conocer la posición del elemento de la lista que pretende ser borrado para poder eliminar otros caracteres (coma o dos puntos precedente). Si hay llamadas implícitas, como se comentó anteriormente, y son computadas, será imposible determinar la posición, lo que puede llevar a la mala formación del mutante.

3. Redeclaraciones de métodos: En C++ podemos encontrarnos con métodos cuya definición no está junto a su declaración dentro de la definición de la clase, lo cual no ocurre en un lenguaje como Java. Esto provoca que los operadores que modifican la declaración de un método necesiten mutar tanto la declaración dentro de la clase como la definición externa para que no se produzca un mutante erróneo.
   Además de este hecho, también hay que considerar que el programador puede haber proveído una declaración de un método, pero no lo ha definido posteriormente. La mutación de tal declaración no tendría sentido y ha de ser evitada.
4. Mutantes sin cambio: Esta situación se da en aquellos operadores que realizan un reemplazo en el código. Por ejemplo, el operador *MNC* o "cambio de nombre de método", cuyo cometido es cambiar el método que se referencia por otro de la misma clase que tenga el mismo tipo y argumentos, debe buscar los métodos por los que puede realizar dicha modificación. Para ello recorrerá los métodos que tiene la clase y, si no se introduce una condición para que no tome como candidato el mismo método cuya invocación se va a mutar, este operador generará un mutante sin cambio al reemplazar el método por sí mismo.

4. Ejemplo de Uso de la Técnica

En esta sección se ilustra el proceso descrito a través de un operador de mutación a nivel de clase concreto, desde la creación del patrón de búsqueda en el AST hasta la generación de los mutantes en un fragmento de código con varias clases de ejemplo.

El Operador de Mutación CDD

El operador *CDD* o "borrado del método destructor" va a ser utilizado en esta sección para ejemplificar cómo podemos combinar diferentes *matchers* a fin de tratar de ajustar la búsqueda a través del AST (según lo prescrito en la definición del operador). No obstante, este procedimiento puede ser utilizado para implementar cualquiera de los 37 operadores que se definieron en [7], así como operadores a otros niveles gracias al uso de un compilador que se centra en las características de este lenguaje. El operador *CDD* tiene como objetivo que el compilador proporcione el destructor por defecto tras eliminar el que provee el programador de la clase. Esto permite comprobar si el destructor que se diseñó es correcto. A la hora de implementar este operador, tendremos en cuenta la siguiente condición: un destructor que no realiza ninguna acción, es decir, que es trivial, no debería ser borrado ya que solo llevaría a crear un mutante equivalente.

Teniendo en cuenta la definición del operador y la condición comentada, se puede construir un *matcher* complejo a través de otros simples para encontrar los

destructores deseados. El *matcher* completo es mostrado en la Figura 1, mientras que los diferentes partes que lo conforman son detalladas a continuación:

```
1 AST_MATCHER(clang::CXXDestructorDecl, isImplicitDestructor){
2       return Node.isImplicit();
3 }
4
5 DeclarationMatcher CDD_Matcher =
6       destructorDecl(
7           unless(
8               isImplicitDestructor()),
9           isDefinition(),
10          has(
11              compoundStmt(
12                      has(
13                          stmt()))))
14      ).bind("CDD");
```

Fig. 1. *Matcher* para el operador de clase *CDD*

```
|-CXXDestructorDecl 0x7932880 parent 0x7931da0 prev 0x79321d0
|                   <line:34:2, line:36:2> ~D 'void (void)'
| `-CompoundStmt 0x7932a18 <line:34:10, line:36:2>
|   `-BinaryOperator 0x79329f0 <line:35:3, col:7> 'class B *'
|                     lvalue '='
|     |-MemberExpr 0x7932988 <col:3> 'class B *'
|                     lvalue ->d 0x7932080
|     | `-CXXThisExpr 0x7932970 <col:3> 'class D *' this
|     `-ImplicitCastExpr 0x79329d8 <col:7> 'class B *'
|                     <NullToPointer>
|       `-IntegerLiteral 0x79329b8 <col:7> 'int' 0
```

Fig. 2. Fragmento de AST con el destructor que toma el patrón del operador *CDD* para la clase "D" del código de la Figura 3.

- Para encontrar el destructor de una clase definido por el usuario:
  - **destructorDecl** (línea 6): Recupera cada uno de los destructores encontrados en las diversas clases.
  - **unless** (línea 7): Es el operador *NO-lógico*. Se trata de ratificar que el destructor no es implícito, tal y como se comentó en el punto 2 de

la Sección 3.3. Para ello, se utiliza el patrón **isImplicitDestructor()** (línea 8), que en este caso ha sido definido por nosotros a fin de poder emplear el método `isImplicit` de la clase `CXXDestructorDecl` (líneas 1-3). Una vez definido el nuevo patrón, es posible emplearlo como cualquier otro *matcher* definido en el DSL.

- A fin de cumplir con la condición impuesta, es decir, no mutar los destructores que sean triviales, se pueden añadir los siguientes patrones:
  - **isDefinition** (línea 9): Para analizar la trivialidad o no de un destructor es necesario buscar la definición del mismo y esto es lo que consigue este patrón. Se garantiza así lo expuesto en el punto 3 de la Sección 3.3 para que no se mute un destructor sin definición.
  - **has(compoundStmt(has(stmt()))** (líneas 10-13): Este patrón compuesto sirve para comprobar si el cuerpo del método tiene en su interior alguna sentencia.

Para finalizar este punto, hay que realizar algunas consideraciones finales:

- El *matcher* **bind** usando la cadena "*CDD*" (línea 14) se especifica como identificador único que nos permite aplicar este patrón cuando se requiere.
- Para que el destructor sea enlazado por el patrón, tienen que cumplirse las tres condiciones impuestas entre los paréntesis de **destructorDecl** (línea 6), que han sido explicadas anteriormente. Por omisión, anidando estas condiciones se aplica el *matcher* **allOf**, que tiene la misma función que el *Y-lógico*.
- Una vez se obtienen los nodos, mediante código C++ y usando las bibliotecas de *Clang*, es posible conseguir un refinamiento mayor mediante la realización de nuevas comprobaciones, como se apuntó previamente en la Sección 3.2. Por último, se puede recuperar la declaración previa para eliminarla también (ver punto 3 de la Sección 3.3).
- Para evitar mutantes duplicados (ver punto 1 de la Sección 3.3), existe el patrón *classTemplateSpecializationDecl* para que no se procese en dos ocasiones el mismo destructor.

Programa de prueba

En la Figura 3 se muestra el código de ejemplo que ha sido diseñado a fin de mostrar cómo funciona el patrón construido. Este código contiene cuatro clases que ilustran diferentes situaciones para el operador.

Los destructores de las clases *A* (línea 10) y *D* (líneas 33, 36-38) deben ser localizados por el patrón para su mutación. La clase *B* (líneas 13-18) carece de destructor explícito, por lo que el patrón no debe tomar el que provee el compilador de forma automática. En el caso de *C*, a pesar de que se ha proporcionado una definición para el destructor (línea 25), esta es trivial ya que no tiene sentencias en el cuerpo del destructor. En la Figura 2 se muestra el nodo que es tomado por el patrón para el destructor de la clase *D*. Esta clase tiene la definición del destructor en una redeclaración del método (líneas 36-38) y, en esa definición, el

```
1  #include <iostream>
2
3  using namespace std;
4
5  class A{
6       public:
7               A(): a(0) {}
8               int a;
9               void asignar_a(int valor) { a = valor;}
10              ~A(){cout << "Valor final: " << a << endl;}
11 };
12
13 class B: public A{
14      public:
15              B(int valor): A(), b(valor) {}
16              int b;
17              int recuperar_b() {return b;}
18 };
19
20 class C{
21      public:
22              C(): c(1.0) {}
23              double c;
24              double recuperar_c() {return c;}
25              ~C() {}
26 };
27
28 class D{
29       public:
30              D(): d(new B(1)) {};
31              B *d;
32              int recuperar_d() {return d->recuperar_b();}
33              ~D();
34 };
35
36 D::~D() {
37      d = 0;
38 }
```

Fig. 3. Código de ejemplo para su uso con el operador *CDD*

método no es trivial. Para que la mutación sea válida en este caso, es necesario el borrado de la declaración y la definición.

Este operador solo generará un mutante por cada localización encontrada: solamente se aplica la operación de borrado del destructor creado por el usuario (en otros operadores, por el contrario, pueden introducirse más de una mutación por cada nodo localizado). De esta manera, una vez visto en el punto anterior las clases cuyo destructor puede ser mutado, el patrón creado para el operador *CDD* generará dos mutantes en el código de la Figura 3. Estos mutantes son exactos al programa original excepto porque los destructores pertinentes han sido comentados en el código:

- Mutante 1:

```
5  class A{
6        public:
7                A(): a(0) {}
8                int a;
9                void asignar_a(int valor) { a = valor;}
10             // ~A(){cout << "Valor final: " << a << endl;}
11 };
```

- Mutante 2:

```
28 class D{
29       public:
30               D(): d(new B(1)) {};
31               B *d;
32               int recuperar_d() {return d->recuperar_b();}
33             // ~D();
34 };
35
36 // D::~D() {
37 //     d = 0;
38 // }
```

5. Conclusiones y Trabajo Futuro

Este artículo presenta una técnica para la implementación de operadores de mutación para C++, un lenguaje al que aún no se había aplicado la prueba de mutaciones de una forma general y que abordase operadores que afectaran a estructuras complejas del lenguaje. Este técnica se fundamenta en el AST generado por el compilador *Clang*, el cual es recorrido gracias a las bibliotecas que proporciona el compilador para la localización de ubicaciones donde es posible introducir las mutaciones. Un ejemplo de la construcción de patrones para determinar esas localizaciones ha sido examinado en el artículo en torno a un operador a nivel de clase, mostrando la versatilidad del método empleado. Este método ha sido de la misma manera aplicado con el resto de operadores de clase

diseñados para C++. La implementación de los operadores de mutación es la base de la construcción de una herramienta de mutaciones ya que nos permite acometer las fases más decisivas de esta técnica de prueba de software: el análisis del código y la generación de los mutantes. Por último, se han enumerado varias situaciones que deben ser tenidas en cuenta a la hora de automatizar la creación de las mutaciones deseadas.

Como trabajo futuro, con el marco de trabajo de mutaciones fundamentado en el enfoque descrito en el artículo, se espera realizar un análisis completo de los operadores relativos a las características de la OO respecto a la distribución de mutantes creados y su utilidad en el propósito de la prueba de mutaciones. Además de las mejoras mostradas para crear mutaciones correctas, se desea poder realizar un estudio en profundidad de los mutantes que genera cada operador para tratar de determinar reglas que eviten la creación de mutantes que no son útiles para el análisis de la técnica. La viabilidad de este mismo modo de implementación para la generación de mutantes en nuevos lenguajes es un campo a explorar.

Referencias

1. Certitude - Functional Qualification System, https://www.synopsys.com/tools/verification/functionalverification/pages/certitude-ds.aspx, Último acceso: 2014.04.14
2. Clang: a C language family frontend for LLVM, http://clang.llvm.org, Último acceso: 2014.04.14
3. Insure++: C/C++ testing tool - parasoft, http://www.parasoft.com/insure, Último acceso: 2014.04.14
4. The LLVM compiler infrastructure, http://llvm.org, Último acceso: 2014.04.14
5. PlexTest ITRegister, http://www.itregister.com.au/products/plextest, Último acceso: 2014.04.14
6. Delgado-Pérez, P., Medina-Bulo, I., Domínguez-Jiménez, J.J.: Analysis of the development process of a mutation testing tool for the C++ language. In: Ninth International Multi-Conference on Computing in the Global Information Technology, ICCGI. Seville, Spain (2014)
7. Delgado-Pérez, P., Medina-Bulo, I., Domínguez-Jiménez, J.J., García-Domínguez, A.: Operadores de mutación a nivel de clase para el lenguaje C++. In: XII Jornadas sobre Programación y Lenguajes, PROLE 2013. Madrid, Spain (2013)
8. Derezinska, A., Kowalski, K.: Object-oriented mutation applied in common intermediate language programs originated from C#. In: Software Testing, Verification and Validation Workshops, IEEE 4th International Conference on. pp. 342–350 (2011)
9. Derezińska, A.: Object-oriented mutation to assess the quality of tests. In: Proceedings of the 29th Conference on EUROMICRO. pp. 417–420. IEEE Computer Society, Belek, Turkey (2003)
10. Derezińska, A.: Quality assessment of mutation operators dedicated for C# programs. In: Kellenberger, P. (ed.) Proceedings of VI International Conference on Quality Software. pp. 227–234. IEEE Computer Society, Beijing (China) (Oct 2006), ISSN 1550-6002

11. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. Software Engineering, IEEE Transactions on 37(5), 649 –678 (Oct 2011)
12. Ma, Y.S., Kwon, Y.R., Offutt, J.: Inter-class mutation operators for java. In: Kawada, S. (ed.) Proceedings of XIII International Symposium on Software Reliability Engineering. pp. 352–363. IEEE Computer Society, Annapolis (Maryland) (Nov 2002), ISSRE 2002
13. Offutt, A.J., Untch, R.H.: Mutation testing for the new century. chap. Mutation 2000: uniting the orthogonal, pp. 34–44. Kluwer Academic Publishers, Norwell, MA, USA (2001), http://dl.acm.org/citation.cfm?id=571305.571314

Pruebas funcionales en programas MapReduce basadas en comportamientos no esperados

Jesús Morán, Claudio de la Riva, Javier Tuya

Departamento de Informática, Universidad de Oviedo, Gijón, España
moranjesus@lsi.uniovi.es, {claudio, tuya}@uniovi.es

Abstract. MapReduce es un paradigma de programación que permite el procesamiento paralelo de grandes cantidades de datos. Los programas MapReduce se suelen ejecutar sobre el framework Hadoop, el cual no garantiza que se ejecuten siempre en las mismas condiciones, pudiendo producir comportamientos no esperados desde el punto de vista de su funcionalidad. En este artículo se analizan y describen diferentes tipos de defectos específicos que pueden estar presentes en programas MapReduce sobre Hadoop y se muestra cómo se pueden derivar casos de prueba que permiten la detección de dichos defectos. Lo anterior se ilustra sobre varios programas de ejemplo.

Keywords: Pruebas de Software, MapReduce, Big Data, Hadoop

1 Introducción

El paradigma de programación MapReduce [1] describe un modelo de programación paralelo basado en "divide y vencerás". Uno de sus ámbitos de aplicación es Big Data, donde se maneja una cantidad masiva de información estructurada y desestructurada.

Un programa MapReduce se suele implementar sobre un framework encargado de la gestión de la paralelización de las diferentes tareas de que se compone, fallos de red, etc. Estos frameworks tienen una forma determinada de ejecutar los programas, como por ejemplo: dividir la entrada en varias partes, enviarlas a distintos computadores o planificar qué computadores realizarán primero el análisis.

Además de los posibles fallos que pueden ocurrir en programas escritos según otros paradigmas, se pueden producir otros específicos de la combinación del paradigma MapReduce y del framework que se utilice. Por ejemplo, puede ocurrir que para una misma entrada al programa unas veces se ejecute correctamente y otras se produzcan fallos; esto ocurre debido a que puede variar el comportamiento de la ejecución del programa, siendo éste un aspecto a considerar cuando se realizan las pruebas.

Durante el desarrollo de este tipo de programas se suele utilizar un entorno muy controlado, donde el programa se ejecuta siempre de forma similar y sin alteraciones de comportamiento. Esto causa que en el entorno de desarrollo se pueden enmascarar

defectos relativos al comportamiento del programa y revelarse posteriormente en producción.

Las contribuciones de este trabajo son (1) identificar posibles fallos funcionales que pueden ser causados por un comportamiento no esperado en un programa MapReduce, y (2) proporcionar unos criterios con los que se puedan realizar las pruebas a estos programas ilustrándolo con programas de ejemplo.

En la sección 2 se resumen los principios del paradigma MapReduce. Posteriormente, la sección 3 contiene una visión general de los aspectos que influyen a nivel de pruebas en programas en MapReduce y un estado del arte. La sección 4 identifica un conjunto de defectos y fallos funcionales que pueden ocurrir desde el punto de vista del comportamiento del programa MapReduce. En la sección 5 se proponen criterios de prueba y se derivan casos de prueba para detectar los defectos, sobre unos programas de ejemplo.

2 Fundamentos MapReduce

En los programas MapReduce se parte de un problema que se divide en varios sub-problemas paralelos que resolver. El modelo de programación consiste principalmente en el desarrollo de una función *Map* y una función *Reduce*, que tienen como entrada y salida conjuntos de pares <clave, valor>.

La función *Map* emite cero o varios pares <clave, valor>, donde la clave representa de forma única a cada sub-problema a resolver y el valor la información que se necesita para resolverlo. La función *Reduce* recibe como entrada pares <clave, lista[valores]> obtenidos del procesamiento de diferentes funciones *Map*, donde se tiene el identificador del sub-problema como clave y una lista de valores que contienen la información necesaria para resolver el sub-problema. Es decir, la función *Map* emite pares <clave, valor>, se agrupan todos los valores emitidos para una misma clave y forman la entrada a la función *Reduce* en forma de pares <clave, lista[valores]>. Finalmente la función *Reduce* produce un resultado en forma de pares <clave, valor>. En cuanto al flujo de ejecución, en primer lugar se ejecutan varias funciones *Map* en paralelo, analizando cada una de ellas una parte distinta de la entrada. Posteriormente se ejecutan varias funciones *Reduce*, también en paralelo. Para las tareas de gestión de problemas de red, uso eficiente de recursos, etc. se suele utilizar un framework, el más utilizado es Hadoop [2].

Además de las funciones *Map* y *Reduce*, puede existir otras funciones, como por ejemplo la función *Combine*, cuyo objetivo es disminuir la cantidad de información que le llega a la función *Reduce* para mejorar el rendimiento. Para ello, la función *Combine* se sitúa entre la salida de la función *Map* y la entrada de la función *Reduce*. Esta función recibe como entrada pares <clave, lista[subconjunto de valores]>, es decir, una clave que identifica a un sub-problema junto con información adicional que necesita ese sub-problema. La función *Combine* realiza un pre-procesamiento del subconjunto de información que le llega y emite pares <clave, valor> que llegan a la función *Reduce* para continuar el ciclo de ejecución. Aunque exista función *Combine*,

Hadoop no garantiza que se vaya a ejecutar, y si se ejecuta, no se sabe con certeza ni cuándo ni con cuántos pares <clave, valor> como entrada [3].

A modo de ejemplo, un programa que calcula la temperatura máxima de un histórico de información de estaciones meteorológicas, podría implementar las tres funciones descritas anteriormente:

- Función *Map*: Recibe pares <año, temperatura>, los valida y los emite como pares <año, temperatura>. Posteriormente, el framework agrupa los pares <año, temperatura> emitidos por la función *Map* en pares <año, lista[temperaturas]>, teniendo cada año una lista de todas con todas sus temperaturas.
- Función *Combine* y *Reduce*: Ambas funciones reciben pares <año, lista[temperaturas]>, es decir, para cada año reciben una lista con sus temperaturas, obtienen la temperatura máxima y emiten un par <año, temperatura máxima>.

En la Fig. 1 se muestra una ejecución en Hadoop del programa donde se tiene como entrada el año 1990 con 1º, 3º y 10º; el año 1991 con 15º y el año 1992 con 1º, es decir, la entrada a la función *Map* son los pares <1900, 1º>, <1900, 3º>, <1900, 10º>, <1991, 15º> y <1992, 1º>.

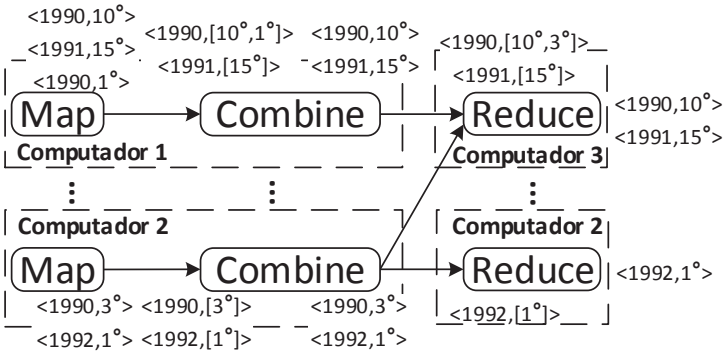


Fig. 1. Ejemplo de ejecución de programa MapReduce

Esta ejecución se realiza en tres computadores, inicialmente y de forma paralela dos de ellos ejecutan las funciones *Map-Combine*. Cada función *Map* envía a la función *Combine* un año y alguna de sus temperaturas, y entre estas, la función *Combine* calcula la temperatura máxima. El espacio total de claves se divide en varios conjuntos disjuntos, que serán asignados a computadores que vayan ejecutar funciones *Reduce*. Cada par <clave, valor> que salga de la función *Map-Combine* se envía al computador que tiene asignada esa clave. El Computador 3, entre otras, tiene asignadas las claves 1990 y 1991, mientras que el Computador 2 tiene asignada la clave 1992. Finalmente se ejecuta cada una de las funciones *Reduce*, estas reciben por cada año todas las temperaturas emitidas por las funciones *Map-Combine* y obtienen la temperatura máxima.

Además de las funciones *Map*, *Combine* y *Reduce*, pueden existir más funciones que interactúen con el programa (como por ejemplo, Partitioner o Sort) así como pro-

gramas MapReduce compuestos de varios *Map-Combine-Reduce* concatenados. En este trabajo se considerarán programas que incluyen *Map*, *Combine* y *Reduce* sin concatenaciones.

3 Trabajo relacionado

Las pruebas en programas MapReduce son clasificadas por Gudipati et al. [4] en las siguientes categorías: "Validation of Pre-Hadoop processing", "Validation of Map Reduce Process", "Validation of Data Extract, and Load into EDW (Enterprise Data Warehouse)", "Validation of Reports", "Performance Testing", "Failover tests" y "Big Data Test infrastructure design". Algunos de estos tipos de pruebas han sido tratados previamente en la literatura e implementados en alguna herramienta como por ejemplo la librería MRUnit [5], la cual permite realizar pruebas unitarias en programas MapReduce especificando qué entrada tiene una función MapReduce y qué salida debería producirse.

El rendimiento de un programa Hadoop proviene tanto de la configuración del cluster como del propio programa [6]. Para probar el rendimiento del cluster Hadoop se pueden utilizar benchmarks como MRBench [7], HiBench [8] o GridMix [9], así como apoyarse en herramientas de monitorización que permitan detectar cuellos de botellas, servidores que son un lastre en el rendimiento, etc. Las pruebas del diseño de la infraestructura son examinadas en Bigtop [10], que consiste en un proyecto para desplegar proyectos Hadoop (ecosistema Hadoop) y probar su interoperabilidad.

En los programas Hadoop, cuando una función *Map*, *Combine* o *Reduce* espera pares <clave, valor> con un determinado tipo de dato y los recibe con otro tipo de dato diferente, puede ocurrir que el programa falle finalizando su ejecución. Dorre et al. [11] realizan una comprobación estática de los tipos de datos de los pares <clave, valor> enviados y recibidos por las funciones *Map*, *Combine* y *Reduce*. Estas pruebas pertenecen a la clasificación "Validation of Map Reduce Process".

Otro tipo de prueba perteneciente al grupo "Validation of Map Reduce Process" es un enfoque de pruebas basado en ejecución simbólica. Estas pruebas requieren que el tester adapte un framework analizando los diferentes caminos del código y unos requerimientos establecidos que debe cumplir el programa, pudiendo crear posteriormente pruebas automatizadas [12].

El enfoque de este artículo se orienta a pruebas funcionales (categoría "Validation of Map Reduce Process"), analizando cómo puede influir el comportamiento de la ejecución de programas MapReduce sobre Hadoop y proponiendo criterios para diseñar las pruebas en fases tempranas del desarrollo del programa.

4 Comportamiento y Defectos en Programas MapReduce

Un programa MapReduce cada vez que se ejecuta en Hadoop puede hacerlo de forma diferente. Por ejemplo, el programa que calcula la temperatura máxima por cada año descrito en la sección 1, puede que una vez se ejecute como se ha representado en Fig. 1 y que otra vez se ejecute como se representa en Fig. 2. En este ejemplo de ejecu-

ción, el comportamiento varía considerablemente, produciéndose los siguientes cambios con respecto a la representada en Fig. 1:

- Hay un tercer computador que analiza en paralelo la entrada al programa MapReduce. Es decir, tres computadores ejecutan funciones *Map*.
- El primer computador analiza dos entradas, mientras el tercero analiza una.
- Una de las funciones *Combine* no recibe el par <1990, [10º, 1º]>, sino que recibe el par <1990, [10º]>.
- El computador 2 envía a la función *Reduce* el par <1990, 3º> antes que el computador 1 le envíe el par <1990, 10º>.
- Una de las funciones *Reduce* en lugar de recibir <1990, [10º, 3º]> recibe <1990, [3º, 10º, 1º]>.
- Se ejecutan tres computadores en paralelo para realizar la funcionalidad *Reduce*, de forma que el computador 3 tiene la clave 1990, el computador 2 la clave 1992 y el computador 1 la clave 1991.

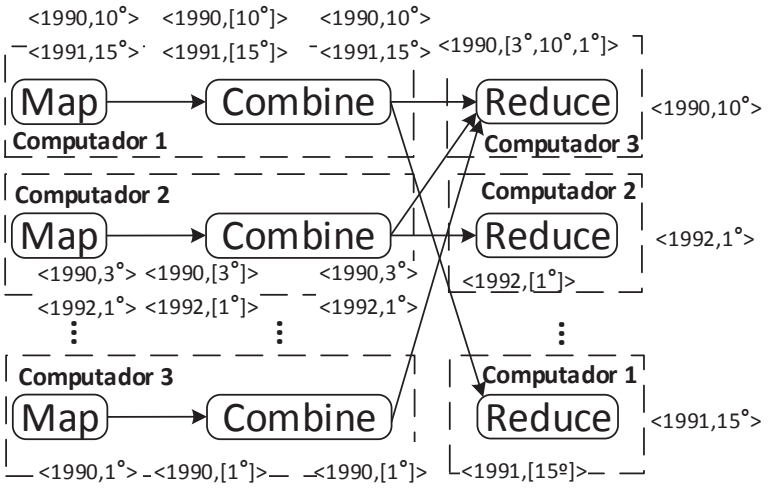


Fig. 2. Ejemplo de otra posible ejecución de programa MapReduce

Que un programa se ejecute de una u otra forma depende de muchos factores: tamaño de cada bloque de entrada que se analiza en paralelo, número de computadores que ejecutan en paralelo el programa, computadores que fallan o que procesan la información a distinta velocidad, etc.

La entrada de las funciones *Combine* y *Reduce* depende del comportamiento que se haya tenido antes, durante y después de la ejecución de la función *Map*, por lo que las funciones *Combine* y *Reduce* son más propensas a tener diferentes comportamientos en cada ejecución. Para las funciones *Combine* y *Reduce* se describen a continuación un conjunto de defectos relativos a un comportamiento inesperado en la ejecución del programa.

4.1 Defectos en la función Combine

La función *Combine* realiza un pre-procesamiento sobre un subconjunto de los pares <clave, valor> emitidos por la función *Map*, y cuya ejecución depende de varios factores, por ejemplo del tamaño de la entrada del programa o del tiempo de procesar entradas y crear pares <clave, valor> en la función *Map*. La función *Combine* se ejecuta unas veces con un subconjunto de información y otras con otro diferente en base a: el tamaño de un buffer que almacena los pares <clave, valor> emitidos por la función *Map*, umbral del buffer a partir del cual se extraen de este los pares <clave, valor> para seguir el ciclo de ejecución, tiempo empleado en vaciar el buffer, etc. Por estas razones la función *Combine* puede ejecutarse con diferentes entradas y en ocasiones produciendo resultados inesperados o enmascarando defectos.

Por otra parte, no todos los programas admiten función *Combine*, pero cuando la función *Reduce* es conmutativa y asociativa [1] se puede definir una función *Combine* con la misma implementación que la función *Reduce*, y viceversa. Cuando la función *Reduce* no es ni conmutativa ni asociativa puede que el programa acepte la función *Combine* teniendo diferente implementación a la función *Reduce*, lo que permite identificar dos diferentes tipos de defectos:

- Defecto por función *Combine* con la misma implementación de *Reduce*
- Defecto por función *Combine* con diferente implementación a *Reduce*.

La realización de pruebas que permitan detectar estos defectos se tratará posteriormente en la sección 5.

4.2 Defectos en la Función Reduce

La función *Reduce* recibe pares <clave, lista[valores]> donde para una misma clave se tienen todos los valores emitidos por las funciones *Map* (y *Combine* en caso que exista). La entrada de la función *Reduce* depende del comportamiento que se tenga en las funciones *Map* y *Combine*.

Pueden ocurrir fallos si los valores en la entrada de la función *Reduce* llegan en distinto orden o si la función *Reduce* está implementada sin tener en cuenta que la función *Combine* puede no ejecutarse o ejecutarse con comportamientos muy diferentes, lo que da lugar a dos tipos de defectos:

- Función *Reduce* dependiente del orden de entrada
- Función *Reduce* no desempeña la funcionalidad de *Combine*

Estos defectos se definen con más detalle a continuación y la realización de pruebas se tratará posteriormente en la sección 5.

Función Reduce dependiente del orden de entrada: En el entorno de producción es difícil garantizar el orden de ejecución de cada función *Map* (las entradas se ejecutan en paralelo, algunas se tienen que volver a procesar porque los computadores pueden fallar, etc.). En cambio, el entorno de desarrollo puede ser un solo computador que no falla durante la prueba y que no necesariamente ejecuta las entradas de forma paralela. Esto implica que es posible que en desarrollo las entradas se ejecuten en el

mismo orden y de forma similar, pero en producción varíe el orden de ejecución, pudiendo por ello darse situaciones no esperadas.

La función *Reduce* podría producir resultados no deseados si los valores en su entrada están en un orden no esperado. Un posible defecto es que la función *Reduce* espere los valores de su entrada ordenados crecientemente y lleguen desordenados.

Función Reduce no desempeña la funcionalidad de Combine: La función *Reduce* tiene que estar implementada bajo la premisa que la información que le llega puede, o no, haberse pre-procesado en la función *Combine*. Si algún par <clave, valor> no pasa por la función *Combine*, la función *Reduce* tiene que procesar correctamente esa entrada, es decir, la funcionalidad de pre-procesamiento que realiza la función *Combine* tiene que ser también realizada por la función *Reduce*, de lo contrario el comportamiento del programa puede ser incorrecto.

5 Pruebas en los Programas MapReduce

Para los defectos presentados en la sección 4, se muestra cómo derivar casos de prueba teniendo en cuenta el comportamiento del programa MapReduce.

En los siguientes apartados, por cada defecto identificado se describe un programa con un defecto junto con una ejecución donde se obtiene la salida deseada. Se proporcionan pautas sobre cómo probarlo y se aplican al programa para obtener casos de prueba. Posteriormente se detalla la salida que se obtendría y el defecto que tiene el programa. Finalmente se indica cómo se ejecutarán las pruebas en un entorno de desarrollo de forma que los resultados sean repetibles.

5.1 Defecto por función Combine con la misma implementación de Reduce

El objetivo es la prueba de un programa que calcula para cada año la temperatura media. La función *Map* recibe años y temperaturas, y emite pares <año, temperatura>. La función *Reduce* recibe pares <año, lista[temperaturas]> y devuelve la temperatura media de cada año.

Si el programa tiene una función *Combine* cuya implementación es la misma que la función *Reduce*, entonces para los años 1900 y 1901, ambos con temperaturas de 1º, 2º, 3º y 4º se tendría que obtener 2.5º de temperatura media. Con esta entrada el programa se podría ejecutar correctamente como se representa en su flujo de ejecución en la figura Fig. 3.

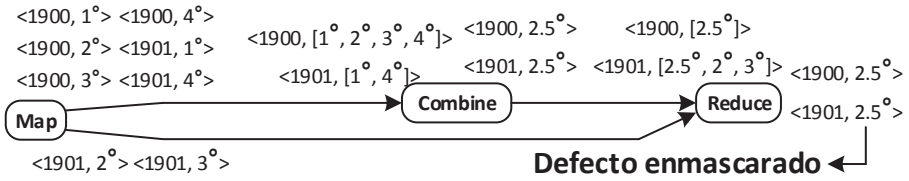


Fig. 3. Posible ejecución de programa que calcula la media de cada año

El objetivo es probar el comportamiento que tiene la función *Combine* cuando recibe diferentes entradas. Para obtener la entrada del caso de prueba, se intenta que no se cumpla la asociatividad entre las funciones *Combine* y *Reduce*. Así, para una clave C se tienen que buscar dos valores que sean emitidos a la función *Combine* y otros dos valores a la función *Reduce*, y para que no se cumpla la asociatividad se tiene que producir diferente resultado al intercambiar un valor de entrada a la función *Combine* por otro de entrada a la función *Reduce*. Esta situación se representa en la figura Fig. 4, donde se intercambian los valores v2 y v3.

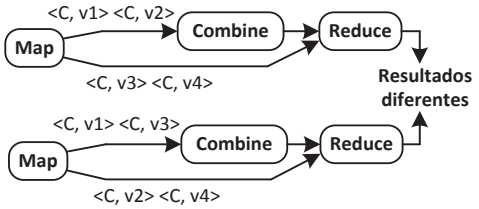


Fig. 4. No cumplimiento de asociatividad entre las funciones Combine y Reduce

La prueba a realizar tiene como entrada al programa los años 1900 y 1901, ambos con temperaturas 1º, 2º, 3º y 4º; donde la función *Combine* recibe como entrada <1900, [1º,2º]> y <1901, [1º,3º]> (se intercambian los valores 2º y 3º). La salida esperada es que los años 1900 y 1901 tengan una temperatura media de 2.5º.

La salida obtenida es que el año 1900 tiene una temperatura media de 2.83º y el año 1901, de 2.66º como se muestra en la figura Fig. 5. El programa tiene un defecto ya que la función que desempeñan *Reduce* y *Combine*, función media, no es asociativa, por tanto la función *Combine* no puede implementar la media.

Para ejecutar las pruebas se necesita un entorno que permita de forma controlada que algunos pares <clave, valor> pasen de la función *Map* a la función *Reduce*; y otros de la función *Map* a la función *Combine* y posteriormente a la función *Reduce*. Este entorno puede consistir en una función que se ejecuta sobre los pares <clave, valor> emitidos por la función *Map*, permitiendo decidir si estos se envían a la función *Combine* o a la función *Reduce*. La ejecución del caso de prueba se debe realizar como se muestra en la figura Fig. 5.

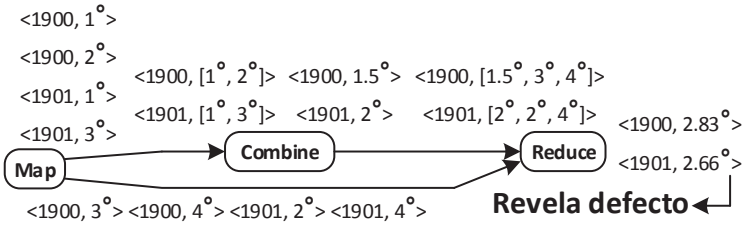


Fig. 5. Ejecución de caso de prueba para programa que calcula la temperatura media de cada año

5.2 Defecto por función Combine con diferente implementación a Reduce

Se va probar un programa que calcula relaciones de seguimiento asimétricas entre contactos de Twitter. Dos personas, por ejemplo Juan y Eva, tienen una relación de seguimiento asimétrica si Juan sigue a Eva pero Eva no sigue a Juan, o viceversa. La función *Map* recibe un identificador de una persona, consigue los contactos a los que sigue, por ejemplo Eva sigue a Juan, y emite pares <Eva, "Sigue: Juan"> y <Juan, "Seguido: Eva">. A la función *Reduce* le llegan pares <Persona, lista[Sigue y seguidos]>, es decir, recibe todos los que le siguen y todos los sigue. Por lo que si una persona sigue y es a la vez seguido por otra, es que no existe relación asimétrica; en otro caso existe relación asimétrica. La implementación correcta es la siguiente:

```
public void reduce (String key, Iterator <String> values) {
  List<String> relationship = new ArrayList<String> (values);
  String contact;
  while (values.hasNext()){
    contact = values.next();
    if (isFollow (contact) && ! existFollower (con-
    tact,values)){
      emit (key, extractName (contact));
      emit (extractName (contact), key);
    }
  }
}
```

Supóngase que al programa se le añade una función *Combine* que al detectar una relación simétrica la filtre y no se la envíe a la función *Reduce*, teniendo un código diferente a la función *Reduce* como el que se muestra a continuación:

```
public void combine (String key, Iterator <String> values){
  List<String> relationship = new ArrayList<String> (values);
  String contact;
  while (values.hasNext()){
    contact = values.next();
    if ( (isFollow (contact) && ! existFollower (con-
    tact,values) ) || (isFollower (contact) && ! existFol-
    low(contact,values)) ){
      emit (key, contact);
    }
  }
}
```

Si se tiene como entrada tres usuarios @1, @2 y @3; donde @1 y @3 siguen a @2; y @2 sigue a @1 y a @3, se debería obtener como resultado que no existen relaciones asimétricas de seguimiento. Una ejecución correcta con estas entradas podría ser la representada en la figura Fig. 6.

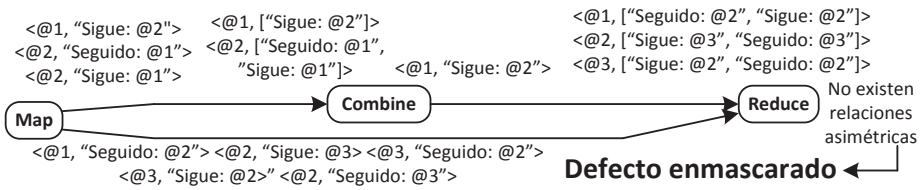


Fig. 6. Posible ejecución de programa que calcula relaciones de seguimiento asimétricas en Twitter

El objetivo es probar el comportamiento que tiene la función *Combine* cuando recibe diferentes entradas. Al igual que en la subsección anterior, para una clave C emitida por la función *Map* se tiene que buscar v1, v2, v3 y v4 que cumplan Fig. 4.

El caso de prueba tiene como entrada tres veces el usuario @1, tres veces @3 y una vez @2, donde además tendrá que ejecutarse la función *Combine* teniendo como entrada <@1, ["Sigue: @2", "Sigue: @2"]> y <@3, ["Sigue: @2", "Seguido: @2"]>. La salida esperada es que no existen relaciones asimétricas entre los usuarios @1, @2 y @3.

La salida obtenida es que entre @1 y @2 no existe relación asimétrica, pero sí existe entre @3 y @2, como se muestra en la figura Fig. 7. El programa tiene un defecto ya que la función *Combine* puede filtrar información aparentemente irrelevante en el subconjunto de pares <clave, valor> que recibe, pero necesaria en el resto de pares <clave, valor> del programa. Por ejemplo, la función *Combine* filtra los pares <@3, "Sigue: @2"> y <@3, "Seguido: @2"> siendo este último necesario para la ejecución correcta de la función *Reduce*.

Un entorno que permita realizar las pruebas de forma controlada tendría que permitir indicar qué pares <clave, valor> van a la función *Combine* y qué pares <clave, valor> van a la función *Reduce*, para finalmente ejecutar el caso de prueba tal como se muestra en la figura Fig. 7. Este entorno podría ser una nueva función que lee los pares <clave, valor> emitidos por la función *Map* y permite decidir si enviarlos a la función *Combine* o a la función *Reduce*.

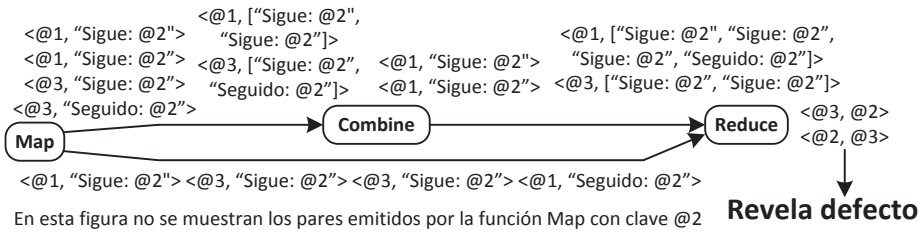


Fig. 7. Ejecución del caso de prueba para programa que calcula relaciones de seguimiento asimétricas en Twitter

5.3 Función Reduce dependiente del orden de entrada

Se va probar un programa que calcula por cada año la temperatura negativa más cercana a cero. La función *Map* recibe años y temperaturas ordenadas decrecientemente; y emite pares <año, temperatura>. La función *Reduce* recibe pares <año, lista[temperaturas]> y debe recorrer todas las temperaturas proporcionando la negativa más cercana a cero por cada año.

Un posible defecto podría ser que la función *Reduce* se implementa suponiendo que los valores de temperaturas le llegan en orden decreciente, devolviendo la primera temperatura menor que cero. El código de la función *Reduce* con este defecto podría ser el siguiente:

```
public void reduce (int key, Iterator <int> values){
  int temp = 0;
  while (values.hasNext()){
    temp = values.next();
    if (temp < 0){
      emit (key, temp);
      break;
    }
  }
}
```

Con esta función *Reduce*, para un programa que tenga como entrada el año 2000 con las temperaturas 20º, 13º, 0º, -2º y -9º se debe tener -2º como temperatura negativa más cercana a cero para el año 2000. Una ejecución correcta del programa con el defecto es la que se representa en la figura Fig. 8.

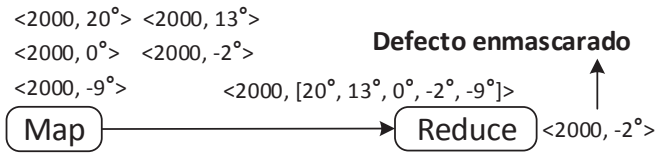


Fig. 8. Posible ejecución de programa que calcula la temperatura negativa más cercana a cero para cada año

La función *Reduce* tiene un defecto ya que si el entorno de desarrollo ejecuta el programa sin realizar paralelismo, la salida siempre sería -2º tal como se muestra en la figura Fig. 8, pero al pasarlo al entorno de producción, podría ser unas veces -2º y otras -9º dependiendo de qué temperatura llegue primero a la entrada de la función *Reduce*.

El objetivo es probar el comportamiento ante diferentes situaciones determinadas por el orden de los valores en la entrada de la función *Reduce*. Por ello se tiene que probar que a la función *Reduce* le lleguen los valores de la entrada en diferente órdenes.

Siguiendo el criterio propuesto, se diseña el siguiente caso de prueba: año 2000 con temperaturas: 20º, 13º, 0º, -2º, -9º (orden decreciente); año 2001 con temperaturas: -15º, -10º, 0º, 3º, 4º (orden creciente); y año 2002 con temperaturas: 0º, -2º, 4º, -5º, 10º (desordenado). La salida esperada es que la temperatura negativa más cercana a cero sea -2º para los años 2000, 2001 y 2002. La salida obtenida es para los años 2000 y 2002, -2º; y para el año 2001, -15º tal y como se muestra en la figura Fig. 9.

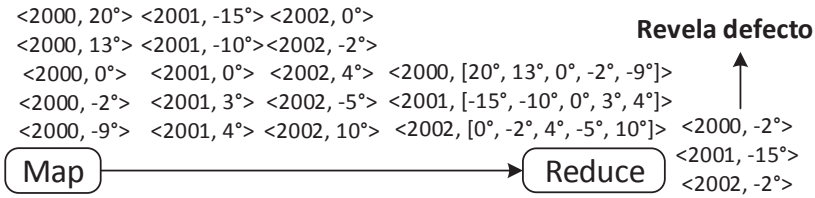


Fig. 9. Ejecución del caso de prueba para programa que calcula la temperatura negativa más cercana a cero para cada año

Para controlar el orden de entrada de los valores en la función *Reduce*, se podría utilizar un entorno de pruebas con un solo computador configurado de forma que la entrada sea ejecutada por las funciones *Map* sin paralelismo, así las pruebas podrían volver a reproducirse con las mismas condiciones.

5.4 Función Reduce no desempeña la funcionalidad de Combine

Se va probar un programa que calcula la temperatura máxima de cada año, y con estas se obtiene la temperatura media de cada rango de 2 años. Cada rango de 2 años irá identificado por el primer año del rango, de forma que la función *Map* emite pares <año, "año temperatura">, <año-1, "año temperatura">. Cada clave se corresponde con el primer año del rango y tiene como valor un año perteneciente al rango y su temperatura. Así las funciones *Combine* y *Reduce* reciben pares <primer año del rango, lista[años del rango junto con su temperatura]>.

Si la función *Combine* calcula la temperatura máxima de cada año y la función *Reduce* la temperatura media, para una entrada con el año 2000 y temperaturas de 0º, 3º, 2º y 1º se debería obtener que en los rangos [1999-2000] y [2000-2001] se tienen 3º como temperatura media de las temperaturas máximas de cada año del rango. La ejecución del programa con esta entrada se muestra en la figura Fig. 10.

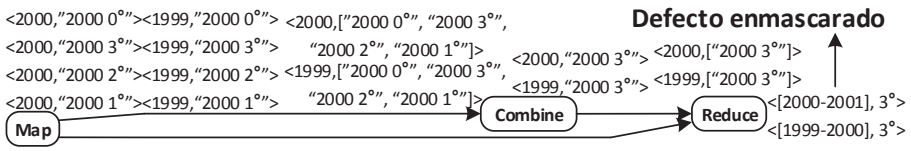


Fig. 10. Posible ejecución de programa que calcula para cada rango de 2 años la temperatura media de las temperaturas máximas de cada año del rango

El objetivo es probar el comportamiento del programa cuando la función *Combine* no se ejecuta y a la función *Reduce* le llegan pares <clave, valor> directamente de la función *Map*. Al igual que en la subsección 5.3 y 5.4, para una clave C emitida por la función *Map* se tiene que buscar v1, v2, v3 y v4 que cumplan la figura Fig. 4 descrita anteriormente.

La entrada del caso de prueba está formada por los años 2000 y 2004, ambos con temperaturas 0º, 3º, 2º y 1º; donde la función *Combine* recibe como entrada <2000, [“2000 3º”, “2000 2º”]> y <2004, [“2004 2º”, “2004 1º”]>. La salida esperada es que en los rangos [1999-2000], [2000-2001], [2003-2004] y [2004-2005] se tenga 3º como temperatura media de la temperatura máxima de cada año del rango.

La salida obtenida es para los rangos [1999-2000] y [2003-3004], 1.5º; para el rango [2000-2001], 1.33º; y para el rango [2004-2005], 1.66º como se muestra en la figura Fig. 11. El programa tiene un defecto ya que la función *Reduce* no implementa la funcionalidad desempeñada en la función *Combine*. La función *Combine* calcula la temperatura máxima de la entrada que recibe, pero como no recibe todas las temperaturas de un año, está calculando el máximo relativo a su entrada, mientras que la función *Reduce* espera que le llegue por cada año una sola temperatura y que sea la máxima.

Un entorno que permita realizar las pruebas de forma controlada tendría que permitir decidir si los pares <clave, valor> emitidos por la función *Map* son entrada a la función *Combine* o a la función *Reduce*. Esto podría hacerse con una función que se sitúe a la salida de la función *Map* y permita decidir si enviar a la función *Combine* o a la función *Reduce* los pares <clave, valor> emitidos. Para el caso de prueba, debe permitir probar lo representado en la figura Fig. 11.

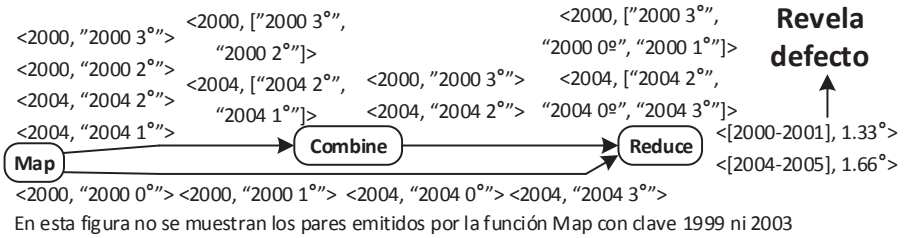


Fig. 11. Ejecución del caso de prueba para programa que calcula para cada rango de 2 años la temperatura media de las temperaturas máximas de cada año del rango

6 Conclusiones

El comportamiento de la ejecución en los programas MapReduce puede causar fallos funcionales, algunos de los cuales pueden provenir por el orden en que le llegan los pares <clave, valor> a la función *Reduce*, por la existencia de una función *Combine* que no debería existir o porque la función *Reduce* no desempeña la funcionalidad de la función *Combine*. Para los defectos mencionados se proporcionan criterios de pruebas y junto a un programa se derivan casos de prueba.

Las líneas fundamentales de trabajo futuro a corto plazo son en primer lugar la ampliación de la identificación y clasificación de defectos dependientes del comportamiento conjunto del programa MapReduce y el framework de ejecución. A continuación será preciso la definición sistemática de los correspondientes criterios de prueba junto con su evaluación en programas más complejos. Por otra parte, se ha comprobado que el control del entorno de pruebas es un elemento esencial para posibilitar la repetibilidad de las pruebas, por lo que una tercera línea deberá abordar estos aspectos junto con la automatización de la realización de las pruebas.

Agradecimientos

Este trabajo ha sido realizado bajo el proyecto de investigación TIN2013-46928-C3-1-R, financiado por el Ministerio de Economía y Competitividad, y fondos FEDER.

Referencias

1. J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In proc. 6th USENIX Sympnosium on Operating Systems Design and Implementation (OSDI), pages 137-149. USENIS, Dec. 2004.
2. Open-source software for reliable, scalable, distributed computing http://hadoop.apache.org/
3. T. White. Hadoop: The definitive guide. O'Reilly, 2012
4. M ahesh Gudipati, Shanthi Rao, Naju D. Mohan and Naveen Kumar Gajja. Big Data: Approach to Overcome Quality Challenges. In Big data: Challenges and opportunities. Infosys Labs Briefings. Vol 11 NO 1 2013.
5. Java library that helps developers unit test Apache Hadoop map reduce jobs. http://mrunit.apache.org/
6. Shrinivas B. Joshi, Apache hadoop performance-tuning methodologies and best practices, Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, April 22-25, 2012, Boston, Massachusetts, USA.
7. Kiyoung Kim, Kyungho Jeon, Hyuck Han, Shin G. Kim, Hyungsoo Jung, and Heon Y. Yeom. Mrbench: A benchmark for mapreduce framework. In ICPADS '08: Proceedings of the 2008 14th IEEE International Conference on Parallel and Distributed Systems, pages 11-18, Washington, DC, USA, 2008. IEEE Computer Society.
8. Shengsheng Huang, Jie Huang, Yan Liu, Lan Yi and Jinquan Da et al. HiBench: A Representative and Comprehensive Hadoop Benchmark Suite.
9. Benchmark for Hadoop clusters. http://hadoop.apache.org/docs/stable1/gridmix.html
10. Development of packaging and tests of the Apache Hadoop ecosystem. http://bigtop.apache.org/
11. J. Dorre, S. Apel, and C. Lengauer, "Static type checking of Hadoop MapReduce programs," in *Proceedings of the second international workshop on MapReduce and its applications.* ACM New York,USA, 2011, pp. 17–24.
12. Christoph Csallner, Leonidas Fegaras y Chengkai Li. New Ideas Track: Testing MapReduce-Style Programs. Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering. Pages 504-507.

Towards Multi-Objective Test Case generation for Variability-Intensive Systems *

Ana B. Sánchez, Sergio Segura and Antonio Ruiz-Cortés

Department of Computer Languages and Systems, University of Seville,
Av Reina Mercedes S/N Seville, Spain
{anabsanchez,sergiosegura,aruiz}@us.es

Abstract. Testing variability-intensive systems is a challenge due to the potentially huge number of derivable configurations. To alleviate this problem, many test case selection and prioritization techniques have been proposed with the aim of reducing the number of configurations to be tested and increasing their effectiveness. However, we found that these approaches do not exploit all available information since they are mainly driven by functional information such as the feature coverage. Furthermore, most of these works are focused on a single-objective perspective (e.g. features coverage), which could not reflect the real scenarios where several goals need to be met (e.g. features coverage and code changes coverage). In this context, we identify an important challenge, to take advantage of all available system information to guide the generation of test cases. As a first step towards a solution, we propose to study all this information with special emphasis on non-functional properties and address the test case generation as a multi-objective problem. Also, we describe some open issues to be explored that we hope have an important impact on future evaluations.

Keywords: Multi-objective test generation, extra-functional attributes

1 Introduction

The ability of a software system to be configurable and adaptable to different environments and users's needs is known as *variability*. Software applications exposing a high degree of variability are usually referred to as *Variability-Intensive Systems (VISs)*. Testing VISs is a challenge due to the potentially huge number of configurations under test. As an example, Debian Wheezy [1] has more than 37,000 packages that can be combined to form millions of different configurations of the operating system. This makes exhaustive testing of VISs infeasible, that is, testing every single configuration is too expensive in general. Furthermore, the software development process usually has to deal with limited available resources and time, which makes that testers must choose the right tests to find

* This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project TAPAS (TIN2012-32273) and the Andalusian Government projects THEOS (TIC-5906) and COPAS (P12-TIC-1867).

as many faults as possible at the right time. Typical approaches for VIS testing use a model-based approach [10], that is, they take an input feature model representing the VIS and return a valid set of features (i.e. configurations) to be tested, i.e. a test suite. In particular, two main strategies have been adopted: test case selection and test case prioritization. *Test case selection* [9, 11] aims at reducing the test space by selecting an effective and manageable subset of configurations to be tested. *Test case prioritization* [5, 12] schedules test cases for execution in an order that attempts to increase their effectiveness at meeting some performance goal, e.g. detecting faults as soon as possible. Both strategies are complementary and are often combined.

Selection and prioritization can be driven by multiple objectives. Typical ones are those based on combinatorial testing where test cases are selected in a way that guarantees that all combinations of t features are tested [11]. The properties of feature models have also been used to select and prioritize test suites in several approaches [6, 12]. Recently, some works have pointed out that not all features are equally relevant and take into account user preferences by assigning weights to the features [8]. Non-functional data such as the number of changes in the code or the number of known defects are recognized as good indicators of the error-proneness of software components [13, 15]. In fact, these data have been used in some works to accelerate the detection of faults by testing first those components with a higher fault propensity [4]. Analogously, non-functional data such as cost or number of users may be good indicators of the impact of certain faults. For instance, the number of downloads of a certain module in an open-source VIS could be used to prioritize test cases running first those exercising the most popular modules. This would allow us to start debugging and correcting faults as soon as possible minimizing the impact on the largest portion of users.

Challenges. Current VIS approaches for test case selection and prioritization are mainly driven by functional information and basic user preferences. However, although this has shown to be effective, these works fail to exploit non-functional information such as the number of code changes or the features size, which are good indicators of their fault propensity and the impact of faults [13]. As a related problem, most contributions for VIS testing use a single objective approach [5, 11], that is, they either aims to maximize a goal (e.g. feature coverage) or minimize another (e.g. suite size) but not both at the same time. Other works [14] combine several goals into a single objective by assigning them weights proportional to their relative importance. While this may be acceptable in certain scenarios, it may be unrealistic in others where users may wish to study the trade-offs among multiple objectives where all of them are equally important. Finally, most works on VIS testing evaluate their approaches in terms of performance (e.g. execution time) using synthetic data [7]. Thus, there is a serious lack of works reporting the results of testing approaches with real data which would provide researchers and practitioners with a better insight into the effectiveness of the testing techniques.

Solution overview. We address these challenges guiding the generation of test cases for VISs using both functional and non-functional attributes in a multi-objective perspective. More specifically, we inspired us on a case study based on a real VIS system presented by some of the authors in [13] to suggest several examples of objective functions using non-functional information. These functions consider the order of the test cases in the suite enabling a combined selection and prioritization approach. We plan to conduct our evaluation using real data extracted from the Drupal framework [13].

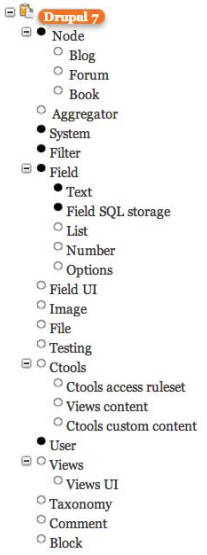
2 Motivating example

We propose to take advantage of the Drupal framework as a motivating VIS to illustrate and evaluate our approach. Drupal is a highly modular open-source web content management framework written in PHP with more than 14,000 modules that can be composed to form valid configurations of the system [3]. More importantly, the Drupal Git repository and the Drupal issue tracking systems are public and inexhaustible sources of valuable functional and non-functional information about the framework and its modules.

Fig 1 depicts the feature model of Drupal v7.23. Nodes in the tree represent features where a feature corresponds to a Drupal module. The model includes the core modules of Drupal, modelled as mandatory features (filled circle icon), plus some optional modules, modelled as optional features (empty circle icon). In addition to the tree, some cross-tree constraints are also depicted. These are of the form `X` requires `Y` which means that configurations including the feature X must also include the feature Y. A *Drupal configuration* is a combination of features consistent with the hierarchical and cross-tree constraints of the model. In total, the Drupal feature model represents 96,768 different Drupal configurations. For more information about how the model was constructed see [13].

In this paper, we propose modelling the non-functional data extracted from the Drupal framework as feature attributes in a feature model. The data were obtained from the Drupal git repository and the Drupal issue tracking system and refer to a period of one year, from September 30th 2012 to September 29th 2013 [13]. In particular, we propose using the following attributes:

- *Feature size*. Number of Lines of Code (LoC) of the source code associated to the feature. The sizes range between 326 LoC and 70,372 LoC (see [13]).
- *Code changes*. Number of commits made by the contributors to the feature in the Drupal git repository. The number of changes ranges from 0 to 43.
- *Single faults*. Number of faults in the feature reported by the users in the Drupal issue tracking system. Faults were collected for two consecutive versions of the framework v7.22 and v7.23. The number of faults found ranges between 0 and 157 (see [13]).
- *Integration faults*. List of features for which integration faults have been reported in the Drupal issue tracking system. For instance, the interaction between the modules `Blog` and `User` caused a fault in the Drupal system according to the case study in [13].



Drupal Cross-Tree Constraints
Forum *Requires* Taxonomy
Forum *Requires* Options
Forum *Requires* Comment
List *Requires* Options
Image *Requires* File
Views content *Requires* Views
Views *Requires* Ctools
Views UI *Requires* Ctools
Taxonomy *Requires* Options

Fig. 1. Drupal v7.23 feature model

– *Tests*. Number of assertions of the features, i.e. a group of methods that check for a condition and return a Boolean, according to the work [13].

3 Basic objective functions

In this section, we present some examples of objective functions based on non-functional attributes. An objective function is a function that represents a goal to optimize. The functions could receive two input parameters, an attributed feature model representing the VIS under test with the non-functional data as feature attributes (afm) and an ordered test suite (ts). As a result, the functions would return a real value measuring the quality of the suite with respect to the optimization goal. These functions would take into account the order of the test cases in the suite. We define the objective functions as follows:

Maximize the number of changes. Changes in the code are likely to introduce faults [15]. Thus, the number of changes in the code of a feature may be a good indicator of its error proneness and could help us to predict faults in later versions of the system. Considering this, we propose an objective function to maximize the number of changes covered by a test suite, giving a higher priority to those test cases including the features with a higher number of changes. Function 1 would calculate the number of code changes, indicated as features attributes, covered by the features included in each test case of the suite ts.

$$nchanges(afm, ts) \tag{1}$$

Maximize the number of faults. Earlier studies show that the detection of faults in an application can be accelerated by testing first those components that showed to be more error-prone in previous versions of the software [4]. Thus, we propose an objective function to maximize the number of faults detected by a test suite giving a higher value to those test cases that detect more faults faster. Function 2 would return the number of faults found in the features of a previous system version that are detected by the test cases of the suite ts.

$$nfaults(afm, ts) \tag{2}$$

Maximize the features size. The size of a feature, in terms of its number of Lines of Code (LoC), can provide a rough idea of the complexity of the feature and its error proneness [13]. Based on this, we propose a novel objective function to maximize the size of the features covered by a test suite, giving priority to those test cases covering higher portions of code faster. Function 3 would return the number of LoC of the features included in each test case of ts.

$$size(afm, ts) \tag{3}$$

Minimize the number of test cases. The number of test assertions in the features of a test case may be used to estimate the test case execution cost. Hence, we propose an objective function to minimize the execution cost of a suite by minimizing the number of assertions of its test cases. Thus, the test cases containing fewer assertions will have a higher priority to be tested. Function 4 would calculate the number of assertions of the features included in the tests cases of ts.

$$ntests(afm, ts) \tag{4}$$

4 Test case generation as a multi-objective problem

In Multi Objective Problems (MOP), algorithms have to optimize two or more objectives (sometimes conflicting). A typical example of a MOP is the Knapsack problem, where the goal is to minimize the weight of a sack while maximising the profit (by placing items into the sack). Considering the objective functions previously mentioned, we propose use them in a multi-objective perspective to drive the generation of effective test suites from a feature model representing a VIS. In a future assessment of our idea, we will have to focus our attention on the possible combinations of the different objective functions. As an example, we could maximize the number of changes covered and the number of faults detected by a suite plus minimize the size of the suite. Another example could be maximize the features size in order to test earlier the more complex configurations and minimize the number of test assertions to reduce the cost of testing.

5 Open issues

Throughout this work, we have identified some open issues to be explored in order to perform our approach, namely:

Combination of objective functions. The definition of new objective functions based on non-functional information could lead to a whole new world of possibilities to help us to optimize the test case generation of VISs. Thus, the key of this issue is to achieve the right combinations of the optimization goals.

Algorithms to resolve the multi-objective problem. In the literature there exist different algorithms to address the multi-objective problem. Among the most popular proposals are the evolutionary algorithms and the mathematical programming algorithms. Thus, the key of this open issue is to choose the best option to resolve the multi-objective problem.

References

1. Debian Wheezy. `http://www.debian.org/releases/wheezy/`, accesed April 2014.
2. D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analyses of feature models 20 years later: A literature review. *Information Systems*, 2010.
3. D. Buytaert. Drupal Framework.`http://www.drupal.org`, accessed in Oct 2013.
4. E. C. P. Cristian Simons. Regression test cases prioritization using failure pursuit sampling. In *ISDA*, 2010.
5. X. Devroey, G. Perrouin, M. Cordy, P.-Y. Schobbens, A. Legay, and P. Heymans. Towards statistical prioritization for software product lines testing. In *International Workshop on Variability Modelling of Software-Intensive*, 2014.
6. J. Ferrer, F. Chicano, and E. Alba. Evolutionary algorithms for the multi-objective test data generation problem. *Software Practice and Experience*, 2012.
7. J. Ferrer, P. Kruse, F. Chicano, and E. Alba. Evolutionary algorithm for prioritized pairwise test data generation. In *Genetic and Evolutionary Computetion Conference (GECCO)*, 2012.
8. M. F. Johansen, O. Haugen, F. Fleurey, A. G. Eldegard, and T. Syversen. Generating better partial covering arrays by modeling weights on sub-product lines. In *International Conference MODELS*, 2012.
9. R. Lopez-Herrejon, F. Chicano, J. Ferrer, A. Egyed, and E. Alba. Multi-objective optimal test suite computation for software product line pairwise testing. In *IEEE International Conference on Software Maintenance*, 2013.
10. S. Oster, A. Wubbeke, G. Engels, and A. Schurr. *A Survey of Model-Based Software Product Lines Testing*, chapter 13, pages 338–381. 2011.
11. G. Perrouin, S. Oster, S. Sen, J. Klein, B. Budry, and Y. le Traon. Pairwise testing for software product lines: comparison of two approaches. *Software Quality Journal*, 2011.
12. A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. A comparison of test case prioritization criteria for software product lines. In *IEEE International Conference on Software Testing, Verification, and Validation*, 2014.
13. A. B. Sánchez, S. Segura, and A. Ruiz-Cortés. The drupal framework: A case study to evaluate variability testing techniques. In *8th International Workshop on Variability Modelling of Software-intensive Systems (VAMOS)*, 2014.
14. S. Wang, S. Ali, and A. Gotlieb. Minimizing test suites in software product lines using weight-based genetic algorithms. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2013.
15. S. Yoo and M. Harman. Regression testing minimisation, selection and prioritisation: A survey. In *Software Testing, Verification and Reliability*, 2012.

Generación automática de casos de prueba mediante siembra automática para WS-BPEL 2.0

Valentín Liñeiro Barea, Antonia Estero Botaro, Antonio García Domínguez, Inmaculada Medina Bulo

Departamento de Ingeniería Informática, Universidad de Cádiz
C/Chile 1, 11002, Cádiz, España
{valentin.lineiro, antonia.estero, antonio.garciadominguez, inmaculada.medina}@uca.es

Resumen. Dada la importancia que en los últimos años están cobrando los servicios web en el ámbito de los procesos de negocio, es imprescindible contar con un soporte de casos de prueba lo suficientemente amplio como para detectar fallos en estos servicios. La automatización de la generación de casos de prueba es importante en este contexto, ya que va a permitir reducir el coste asociado a las tareas de prueba. En este trabajo se aplica por primera vez a composiciones WS-BPEL 2.0 la siembra automática, que combina la generación aleatoria de casos de prueba con información adicional de las constantes del programa. Además, se define una optimización de esta técnica para evitar la generación de conjuntos grandes de casos de prueba.

Palabras clave: *generación automática de casos de prueba, siembra automática, servicios web, WS-BPEL*

1. Introducción

En los procesos de negocio, los servicios web (WS) van a jugar un papel fundamental a la hora de gestionarlos. Casos reales nos podemos encontrar muchos como, por ejemplo, la reserva de un viaje con todo lo que puede conllevar, o la aprobación de un préstamo teniendo en cuenta el riesgo que implica la operación. Por tanto, es fundamental que tanto los WS como las composiciones que los emplean sean robustos.

La aplicación de técnicas de prueba a los WS y a sus composiciones pueden contribuir a su robustez. Un aspecto fundamental de la prueba de software es la selección de los casos de prueba a utilizar en la prueba de un programa. Dado que el dominio de entrada de un programa puede ser muy grande, se necesita seleccionar un pequeño conjunto de casos de prueba representativo del dominio de entrada completo. Una gran parte de la investigación realizada en el campo de la prueba de software se ha centrado en la generación de casos de prueba atendiendo a diversos criterios. Podemos distinguir así los métodos

estructurales, que generan casos de prueba que deben cubrir ciertos elementos de la estructura del programa; los métodos basados en fallos, que miden la capacidad de un conjunto de casos de prueba para encontrar fallos en un programa, y los métodos basados en errores, que generan casos de prueba que comprueban el programa en ciertos puntos más propensos a errores.

La automatización de la prueba de software es un aspecto crucial de esta [3], siendo especialmente importante la automatización de la generación de casos de prueba, ya que va a permitir reducir el coste asociado a las tareas de prueba. Las técnicas de generación automática de casos de prueba pueden emplear el código del programa, una especificación de la entrada del mismo para realizar su trabajo o ambas a la vez. Este trabajo versará sobre esta última categoría.

Este artículo tiene como principal objetivo la definición e implementación de una técnica de generación de casos de prueba mixta, la siembra automática, para WS-BPEL 2.0, que permitirá obtener casos de prueba empleando tanto el conocimiento residente en la entrada del programa como en su propio código.

WS-BPEL 2.0 [5] es un lenguaje basado en XML que permite reunir en un solo servicio varios WS existentes, obteniéndose así una *composición de servicios*. Este lenguaje soporta la invocación de WS externos tanto de forma síncrona como asíncrona, así como la concurrencia, lo que hace más complejo el proceso de prueba por las características inherentes que ésta aporta.

El resto del artículo se estructura de la siguiente forma. En la Sección 2 se especifica en qué consiste la siembra automática. La Sección 3 presenta una serie de trabajos relacionados y, por último, en la Sección 4 se presentan las conclusiones y el trabajo futuro.

2. Siembra automática

La siembra automática [1,2,4] es una técnica de automatización de la generación de casos de prueba que extiende la técnica de generación aleatoria utilizando las constantes existentes dentro del programa a probar. Esta técnica puede encuadrarse [6] dentro de las técnicas de automatización mixtas, ya que combina una técnica de automatización basada en la interfaz, como la aleatoria, con el uso de las constantes existentes en el programa, lo que la encuadra dentro de las técnicas basadas en la implementación.

A continuación se describe el proceso que se propone para generar los casos de prueba aplicando la siembra automática.

1. Obtener las constantes de la composición WS-BPEL.
2. Obtener las variables que forman parte de la especificación de los casos de prueba.
3. Asignar a cada variable las constantes que son compatibles con ella en el sentido de los tipos de datos.
4. Generar un conjunto de casos de prueba aleatorio inicial.
5. Por cada caso de prueba, realizar una modificación sustituyendo en la variable seleccionada el valor de constante elegido.

Listado 1.1. Fragmento de ejemplo para la aplicación de la siembra automática

```
<if name="ifPrincipal">
   <condition>$temp.part/ns0:estado = 'ABIERTO'</condition>
   <while>
      <condition>$input.part/ns1:datos/ns1:temperatura > 40
      and $input.part/ns1:datos/ns1:presion > 400</condition>
      ...
   </while>
   <assign>
      <copy>
         <from>'CERRADO'</from>
         <to>$temp.part/ns0:estado</to>
      </copy>
   </assign>
</if>
```

Listado 1.2. Especificación de variables correspondiente al listado 1.1

```
typedef string (values={"ABIERTO", "CERRADO"}) Estado;
typedef int (min=0, max=2000) Magnitud;
Estado estado;
Magnitud temperatura, presion;
```

Supongamos que tenemos una colección de tipos básicos T en el lenguaje en el que están expresados los programas. Entonces, tendremos, para cada tipo $i \in T$, una colección V_i de variables del tipo y una colección C_i de constantes compatibles con el tipo, y por tanto, con el conjunto V_i.

El número de casos de prueba que se genera, p, se puede obtener a partir de la siguiente fórmula: $p = \sum_{i=1}^{|T|} |V_i| \cdot |C_i|$. Dado t, índice dentro de la colección de un tipo cualquiera que pertenece a T, i, índice dentro de la colección de una variable que pertenece a V_t y j, índice dentro de la colección de una constante que pertenece a C_t, podremos localizar el caso de prueba donde se produce el cambio de la siguiente forma: $c_k : V_{ti} \leftarrow C_{tj},\ k = |T| \cdot (t-1) + |V_t| \cdot (i-1) + j$.

Ejemplo 1 Los listados 1.1 y 1.2 muestran un fragmento de código y su correspondiente especificación de variables. En este caso, tendríamos lo siguiente:

- Colección de tipos: $T = \{string, int\}$. En la fórmula para calcular p, se indexan por el orden establecido ($string = 1$, $int = 2$).
- Colecciones de variables por tipo: $V_1 = \{estado\}$ y $V_2 = \{temperatura, presion\}$.
- Colecciones de constantes por tipo: $C_1 = \{ABIERTO, CERRADO\}$ y $C_2 = \{40, 400\}$.

Dado que $|T| = 2$, $|V_1| = 1$, $|V_2| = 2$, $|C_1| = 2$ y $|C_2| = 2$, el número de casos de prueba a generar sería: $p = \sum_{i=1}^{|T|} |V_i| \cdot |C_i| = \sum_{i=1}^{2} |V_i| \cdot |C_i| = 1 \cdot 2 + 2 \cdot 2 = 6$.

En la tabla 1 podemos ver el conjunto de casos de prueba generado. Los valores de las variables con puntos suspensivos representan valores aleatorios cualesquiera dentro del dominio de cada variable.

Tabla 1. Conjunto de casos de prueba generado para el fragmento del listado 1.1.

	1	2	3	4	5	6
`estado`	`ABIERTO`	`CERRADO`	...	...	...	...
`temperatura`	...	...	`40`	`400`	...	...
`presion`	...	...	...	...	`40`	`400`

Optimización

Dependiendo del programa para el que estemos generando el conjunto de casos de prueba, puede haber muchas variables a las que se le pueda asignar un conjunto de constantes compatibles. Esto implica que $|V_i|$ y $|C_i|$ serán grandes, y por tanto, p también. En entornos donde la ejecución de cada caso de prueba sea costosa en términos de tiempo o espacio, como ocurre con WS-BPEL 2.0, esto supone un problema para aplicar la siembra automática. Para paliarlo, se propone una optimización de la técnica básica que se basa en ampliar la restricción de compatibilidad de las variables y las constantes.

Para cada tipo $t \in T$, cada variable $v_t \in V_t$ se relaciona con todas las constantes que pertenecen a C_t. Esto hace que se generen casos de prueba que no son significativos, pues únicamente nos interesan aquellos donde las variables toman las constantes con los que están relacionadas, sea por comparación o por asignación, dentro del programa. Para resolver este problema, planteamos una optimización que consiste en obligar a que la variable esté relacionada con la constante dentro del programa para ser compatible con ella. Para poder aplicar la optimización, se necesita realizar un mapeo entre las variables de la especificación y las variables propias de la composición.

Con esta nueva restricción, obtenemos: $p' = \sum_{i=1}^{|T|} \sum_{j=1}^{|V_i|} |C_{ij}|$. En esta expresión aparece $|C_{ij}|$, que es el cardinal de la colección de constantes de un tipo dado i compatibles y relacionadas con una variable j. Dado que $|C_{ij}| \leq |C_i|$, entonces tendremos que $p' \leq p$.

La diferencia entre p' y p depende obviamente del programa donde se aplique la técnica. La eficacia de la optimización puede medirse, para cada tipo i, como $|C_i| - |C_{ij}|$. Cuando aplicamos la optimización, la forma de localizar los casos de prueba difiere, siendo: $c_k : V_{ti} \leftarrow C_{ij},\ k = |T| \cdot (t-1) + |C_{ij}| \cdot (i-1) + j$.

Ejemplo 2 Utilizaremos de nuevo el fragmento del listado 1.1 y la especificación de variables del listado 1.2 para generar el conjunto de casos de prueba aplicando la optimización que emplea el mapeo existente en el listado 1.3.

Listado 1.3. Mapeo correspondiente a los listados 1.1 y 1.2

```
...
<tes:template>
   <ns0:estado>$estado</ns0:estado>
</tes:template>
...
<tes:template>
   <ns1:datos>
      <ns1:temperatura>$temperatura></ns1:temperatura>
      <ns1:presion>$presion</ns1:presion>
   </ns1:datos>
</tes:template>
...
```

Tabla 2. Conjunto de casos de prueba generado aplicando optimización.

	1	2	3	4
estado	ABIERTO	CERRADO	...	...
temperatura	...	...	40	...
presion	...	...	...	400

Para este ejemplo, las colecciones T, V_1 y V_2 son las mismas que en Ejemplo 1. Pero al aplicar la optimización, las colecciones C_1 y C_2 no se toman al completo, sino que solo se consideran las subcolecciones de constantes que se relacionan con cada variable: $C_{11} = \{ABIERTO, CERRADO\}$, $C_{21} = \{40\}$ y $C_{22} = \{400\}$.

Sabiendo que $|T| = 2$, $|V_1| = 1$, $|V_2| = 2$, $|C_{11}| = 2$, $|C_{21}| = 1$ y $|C_{22}| = 1$, el número de casos de prueba sería: $p' = \sum_{i=1}^{|T|} \sum_{j=1}^{|V_i|} |C_{ij}| = \sum_{i=1}^{2} \sum_{j=0}^{|V_i|} |C_{ij}| = \sum_{j=1}^{1} |C_{1j}| + \sum_{j=1}^{2} |C_{2j}| = 2 + 1 + 1 = 4$. Es decir, en este caso sólo necesitamos generar 4 casos de prueba frente a los 6 que generábamos sin aplicar la optimización. La tabla 2 muestra el conjunto de casos de prueba generado, al comparar con la tabla 1 vemos que desaparecen los casos de prueba 4 y 5, que no son significativos.

3. Trabajos Relacionados

En esta sección se describen los trabajos que aparecen en la literatura relacionados con la siembra automática.

Alshraideh et al. [2] consideran el problema de generar datos de prueba que cubran ramas de un programa que dependen de predicados de cadenas. Definen una técnica de generación de casos de prueba basada en búsqueda que emplea operadores de búsqueda dependientes del programa dado, utilizando las cadenas que aparecen en el programa de forma que los operadores de búsqueda centran esta búsqueda en la región de las cadenas literales.

Alshahwan et al. [1] aplican el enfoque propuesto por Alshraideh et al. [2] a la generación de casos de prueba para aplicaciones web, introduciendo el proceso de siembra de constantes estáticas (SCS), el cual consiste en tomar las constantes de los programas como valores iniciales para generar los casos de prueba.

Fraser et al. [4] estudian las estrategias de siembra en diferentes contextos: siembra de constantes extraídas del código a lo largo de todo el proceso de búsqueda (población inicial, operadores de mutación, ...), estrategias para mejorar la población inicial de la búsqueda y reutilización de soluciones previas (casos de prueba generados previamente o creados a mano) para sembrar la población inicial de la búsqueda. El trabajo compara estas estrategias para programas escritos en Java y concluye que tienen un impacto positivo en la búsqueda evolutiva.

El trabajo que se presenta en este artículo aplica la siembra automática a composiciones WS-BPEL e introduce una técnica de optimización del número de casos de prueba generados

4. Conclusiones y Trabajo Futuro

El trabajo presenta la técnica de siembra automática y su aplicación al lenguaje WS-BPEL 2.0. Además se presenta la optimización realizada para reducir el número de casos de prueba que se genera.

El trabajo futuro se centrará principalmente en realizar un estudio experimental que compare la técnica presentada, tanto en su versión convencional como optimizada, con otras técnicas de generación de casos de prueba.

Agradecimientos. Este trabajo ha sido financiado por el proyecto MoDSOA (TIN 2011-27242) del Programa Nacional de I+D+i del Ministerio de Ciencia e Innovación.

Referencias

1. Alshahwan, N., Harman, M.: Automated web application testing using search based software engineering. In: ASE'11: 26th IEEE/ACM International Conference on Automated Software Engineering. pp. 3–12 (2011)
2. Alshraideh, M., Bottaci, L.: Search-based software test data generation for string data using program-specific search operators. Software Testing, Verification and Reliability 16(3), 175–203 (2006)
3. Bertolino, A.: Software testing research: Achievements, challenges, dreams. In: FoSE'07: Future of Software Engineering 2007. pp. 85–103. IEEE Computer Society (2007)
4. Fraser, G., Arcuri, A.: The seed is strong: Seeding strategies in search-based software testing. In: ICST'12: IEEE Fifth International Conference on Software Testing, Verification and Validation. pp. 121–130. IEEE Computer Society (2012)
5. OASIS: Web Services Business Process Execution Language 2.0 (2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
6. Tuya, J., Román, I.R., Cosín, J.D.: Técnicas cuantitativas para la gestión en la ingeniería del software, chap. 14, pp. 310–311. NetBiblo (2007)

Multi-dimensional Criteria for Testing Web Services Transactions

Rubén Casado[1], Muhammad Younas[2], Javier Tuya[1]

[a]Departamento de Informática, Universidad de Oviedo, Spain
{rcasado, tuya}@uniovi.es
[b]Department of Computing and Communication Technologies, Oxford Brookes University, UK
m.younas@brookes.ac.uk

Abstract. Web services (WS) transactions are important in order to reliably compose web services and to ensure that their execution is consistent and correct. This paper addresses the key issue of testing WS transactions by proposing multi-dimensional criteria that provide tester with flexibility of adjusting the method in terms of test efforts and effectiveness. The criteria have been designed, implemented and evaluated through a case study. This work has been published in Journal of Computer and Systems Sciences[1].

Keywords: web services, transactions, testing, classification-tree

1 Introduction

Transactions in Web Services (WS) are different from traditional transactions which support ACID (Atomicity, Consistency, Isolation and Durability) properties. The management of transactional activities complicates the business logic of WS as their execution requires careful coordination, accounting for fault-tolerance, correct process termination and cancelation, without undesirable consequences at any stage of the execution. Current research proposes various solutions for testing the (non-transactional) WS but testing WS transactions has not been researched yet [1].

2 Contributions

This work presents multi-dimensional criteria for testing the WS transactions. The criteria define three dimensions; *Level*, *Feature* and *Depth*. The *Level* dimension defines the granularity level of testing, i.e., testing WS transactions at different levels such as activity (web service) level [2], nested subtransaction or overall transaction level. The *Feature* defines the source used to identify the situations to be tested (test conditions). For example, the relationships between activities, the execution flow of activities or the data elements shared by the activities of a WS-transaction. The *Depth* is related to test coverage items, designing of the test cases, and the cost-benefit trade-off of testing the WS transactions. Contributions of this work are as follows:

1. **A method for designing test cases for WS transactions**. We present systematic multi-dimensional approach to identify test conditions and test coverage items by

[1] R. Casado, M. Younas, and J. Tuya, "Multi-dimensional Criteria for Testing Web Services Transactions " *Journal of Computer and Systems Sciences* vol. 79, pp. 1057-1076, 2013. DOI: http://dx.doi.org/10.1016/j.jcss.2013.01.020. Impact Factor (IF): 1.0, 5-year IF: 1.106. Journal ranked as 34 of 100 in the Computer Science, Theory & Methods category.

analysing the existing dependencies between the involved activities using the Classification-Tree technique. An abstract dependency model is introduced.
2. **A family of test criteria**. New test design concepts (*Primitive TCI, Combined TCI, TCI selection, TCI combination, strong level, weak level.*) are introduced and, based on them, different criteria have been proposed to adjust the test efforts.
3. **Evaluation**. We have evaluated the proposed criteria using The Travel Agency case study which has been widely discussed in the literature [3][4]. The test suites generated using the different test criteria were executed in mutated versions of the program.

3 Results and conclusions

We evaluated the proposed criteria by taking into account the following research questions and using a mutation-based approach:

- **RQ1:** Effectiveness of the proposed criteria in detecting failures in WS Transactions.
- **RQ2:** Usefulness of the proposed criteria useful in adjusting the test efforts and providing a trade-off in terms of cost-benefit.
- **RQ3:** Resiliency of the proposed criteria to different types of defects or failures.

All test suites achieve a mutation score greater than 65% and some score greater than 80%. Therefore, the proposed criteria are indeed effective thus meeting the RQ1. With regard to the RQ2, different level and combination criteria lead to different test efforts measured in terms of the number of test cases generated. The benefit (killed mutants) also differs in different test suites. In relation to the RQ3 the results show that the type of fault influences the effectiveness of the testing criteria.
As conclusion, evaluation results showed that the proposed criteria have the potential to design effective test cases for WS transactions and to allow the tester to adjust the method in terms of its effectiveness, test effort and cost-benefit analysis. Further the design of the test cases is automatically generated in order to meet the requirements of the distinguishing characteristics of WS transactions. It reduces the cost of the test design and also improves its effectiveness. It allows for adjusting the intensity of the test process by taking into account the time and effort.

4 References

[1] M. Bozkurt, M. Harman, and Y. Hassoun, “Testing Web Services: A survey,” Department of Computer Science, King’s College London, Technical Report TR-10-01, 2010.

[2] R. Casado, J. Tuya, and M. Younas, “Evaluating the effectiveness of the abstract transaction model in testing Web services transactions,” *Concurr. Comput. Pract. Exp.*, p. in press, 2012.

[3] L. Bocchi, C. Laneve, and G. Zavattaro, “A Calculus for Long-Running Transactions Formal Methods for Open Object-Based Distributed Systems,” vol. 2884, E. Najm, U. Nestmann, and P. Stevens, Eds. Springer Berlin / Heidelberg, 2003, pp. 124–138.

[4] J. Jiang, G. Yang, Y. Wu, and M. Shi, “CovaTM: a transaction model for cooperative applications,” *Proceedings of the 2002 ACM symposium on Applied computing*. ACM, Madrid, Spain, pp. 329–335, 2002.

PTAC: Una herramienta para testing pasivo de sistemas con comunicaciones asíncronas

Mª Azahara Camacho Magriñán, Inmaculada Medina Bulo y Mercedes G. Merayo

Grupo UCASE de Ingeniería del Software, Departamento de Ingeniería Informática
Universidad de Cádiz, Escuela Superior de Ingeniería, Cádiz, España,
azahara.camacmagri@alum.uca.es, inmaculada.medina@uca.es

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Madrid, España,
mgmerayo@fdi.ucm.es

Resumen Las técnicas de pruebas pasivas nos ofrecen la ventaja de poder testear sistemas con los cuales no es posible la interacción directa. Actualmente, la gran mayoría de las propuestas basadas en este concepto asumen que las trazas analizadas reflejan el comportamiento real del sistema, sin tener en cuenta el canal a través del cual han sido observadas. Si la comunicación es asíncrona, la traza observada puede no corresponderse con la producida durante la ejecución del sistema. La necesidad de considerar este tipo de aspectos, llevó a la propuesta de una nueva metodología de pruebas pasivas centrada en canales asíncronos First Input First Output (FIFO). En este artículo se presenta PTAC, una herramienta software que automatiza la aplicación de esta técnica, permitiendo el análisis del comportamiento de los sistemas, mediante la validación de las diferentes propiedades que lo caracterizan a través de trazas reales.

Palabras claves: Prueba de software, pruebas pasivas, sistemas asíncronos.

1. Introducción

Dentro de las diferentes técnicas de validación de software, las técnicas de testing son las más aplicadas en la industria. Existe una clasificación general de dichas técnicas que las divide en pruebas activas y en pruebas pasivas dependiendo de si es o no posible interactuar con el sistema bajo testeo. Gran parte de los sistemas pueden ser evaluados mediante la aplicación de técnicas de pruebas activas, pero existen otros que únicamente pueden ser testeados mediante metodologías pasivas [5].

La mayoría de las metodologías de pruebas pasivas propuestas asumen que las trazas analizadas muestran el comportamiento real del sistema. Sin embargo, es posible que esas trazas hayan sido observadas a través de un canal asíncrono

[6] que haga que la observación del sistema no se corresponda con la ejecución real, ya que algunas acciones pueden aparecer retrasadas en la traza observada con respecto a la traza original.

La necesidad de considerar este tipo de entornos asíncronos y sus peculiaridades llevó a la propuesta de una nueva metodología de pruebas pasivas centrada en canales First Input First Output (FIFO) [4]. Esta propuesta tiene como objetivo la evaluación en tiempo real del cumplimiento, por parte de estos sistemas, de propiedades que describen el comportamiento que deben mostrar.

En este artículo se presenta PTAC (Passive Testing with Asynchronous Communications), una herramienta que implementa dicha técnica. Específicamente, permite el análisis del comportamiento de los sistemas considerados, respecto a diferentes propiedades establecidas previamente.

El resto del artículo presenta la siguiente estructura. En la Sección 2 se introducen los fundamentos teóricos de la técnica implementada. La Sección 3 describe las funcionalidades de la herramienta desarrollada, así como el caso de estudio aplicado para su evaluación. Finalmente, en la Sección 4 se presentan las conclusiones y algunas líneas de trabajo futuro.

2. Metodología de la Técnica

La novedad que aporta la técnica propuesta en [4] es el tratamiento y consideración de las diferencias existentes entre el comportamiento real de un sistema y el percibido por el testeador a través de un canal asíncrono. La principal característica que reflejan estas observaciones son retrasos temporales que provocan una alteración en la secuencia de las acciones sucedidas durante la ejecución del sistema.

La técnica en la que se basa PTAC se centra en un algoritmo que genera un autómata que aceptará las secuencias observadas en un sistema asíncrono, que no satisfagan una propiedad previamente definida. Las propiedades consideradas en el marco propuesto indican las acciones válidas que pueden producirse en el sistema, si este ha generado una traza determinada.

La construcción del autómata final se realiza en dos fases:

- Se genera un grafo inicial con todas las secuencias de acciones que pueden observarse si el sistema ha producido la traza considerada en la propiedad.
- El autómata es adaptado para considerar todas las posibles acciones que pueden haberse producido antes y después de generarse la traza en el sistema, incluyendo aquellas que puedan darse debido a la naturaleza FIFO del entorno.

Si la traza observada se ha generado debido a la producción de una secuencia de acciones por parte del sistema que no cumple la propiedad asociada al autómata construido, dicha traza será aceptada por el autómata indicando que no se ajusta a los requisitos presentados. En el caso de que se ajuste a la propiedad o su observación sea una combinación de acciones compatible con dicha propiedad, se continuará la evaluación de la traza.

3. Descripción de la herramienta PTAC

PTAC es una aplicación que incluye una interfaz gráfica desarrollada en Java con la ayuda de los componentes ofrecidos por SWT [3] y se suministra de la API de Java en código abierto de GraphViz para poder representar los gráficos generados a partir de la aplicación del algoritmo con el lenguaje DOT [1]. Las principales funcionalidades que ofrece el sistema son:

- Gestión de propiedades: Permite al usuario la definición, modificación y eliminación de las propiedades de un sistema.
- Generación de autómata: Mediante la aplicación del algoritmo descrito en la Sección 2, se obtiene el grafo correspondiente al autómata asociado a la propiedad considerada.
- Sistema de testeo: Permite al usuario comprobar la validez de trazas observadas frente a una propiedad específica.

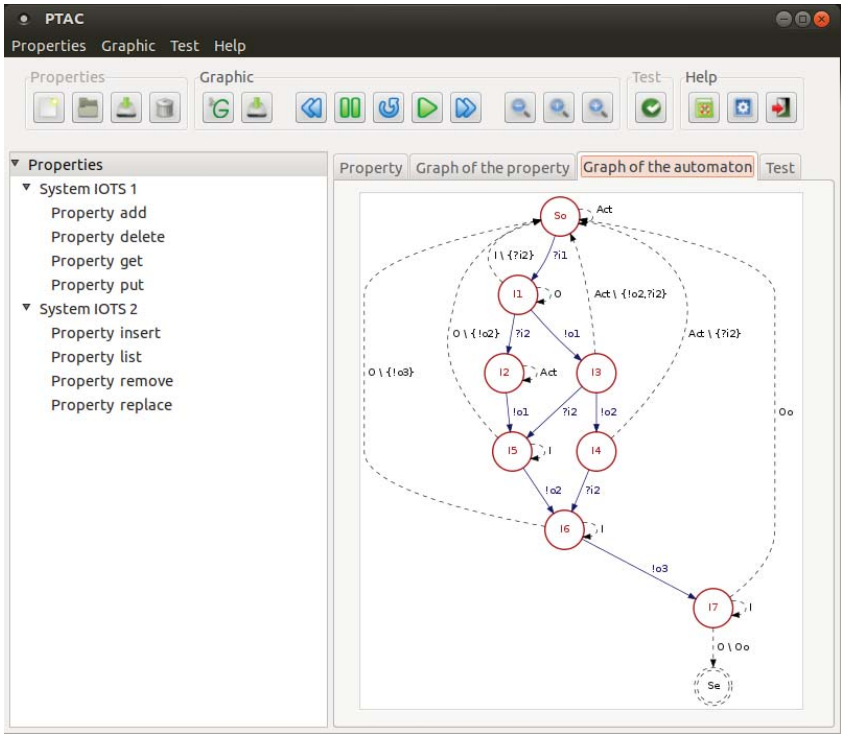


Fig. 1: Muestra de la generación de un autómata con PTAC

Esta aplicación ha sido desarrollada para su ejecución en cualquier máquina con procesadores de 32 ó 64 bits y existen versiones para sistemas operativos *Windows* y *Linux*.

PTAC ofrece a los usuarios la posibilidad de llevar a cabo pruebas de un gran nivel de complejidad, sin necesidad de poseer unos amplios conocimientos. Conocer las propiedades del sistema que quiere testear y saber adaptarlas al formato aceptado por PTAC serán los únicos requisitos que deberá cumplir para realizar un proceso de testing adecuado.

Entre las diferentes ventajas que ofrece esta herramienta se encuentran la reducción del coste temporal de dicho proceso gracias a la automatización proporcionada por PTAC, la reducción significativa de posibles errores humanos que pueden cometerse a la hora de llevar a cabo dicha aplicación de forma manual y la difusión de la técnica implementada a través de una interfaz cómoda para el usuario.

Un ejemplo de su utilidad es la aplicación de PTAC en el caso de estudio del protocolo de red HyperText Transfer Protocol (HTTP) [2]. En el experimento se definieron diferentes propiedades que deben cumplirse en las comunicaciones del protocolo y se capturaron trazas reales para comprobarlas. Estas trazas fueron mutadas y sometidas a testeo en PTAC, obteniendo como resultado la detección de los errores inyectados en las trazas.

4. Conclusiones y Trabajo Futuro

En este trabajo se ha presentado PTAC, una aplicación que incluye una interfaz gráfica para realizar testing pasivo en base a la propuesta presentada en [4], para sistemas que presentan asincronía en los canales de observación.

Como trabajo futuro, se pretende continuar con el desarrollo y perfeccionamiento de cada una de las funcionalidades que posee PTAC, con la intención de hacer lo más manejable posible el proceso de testing. También se ha planteado la posibilidad de crear un modo estudiante incluido dentro de la misma interfaz, más detallado a la hora de mostrar resultados y procesos intermedios, haciendo posible la divulgación del testing de software entre los estudiantes de Informática.

Referencias

1. DOT Manual, http://www.graphviz.org/cgi-bin/man?dot
2. Especificación del protocolo HTTP. http://www.ietf.org/rfc/rfc2616.txt
3. SWT, The Standard Widget Toolkit, http://www.eclipse.org/swt/
4. Hierons, R.M., Merayo, M.G., Núñez, M.: Passive testing with asynchronous communications. In: IFIP 33rd Int. Conf. on Formal Techniques for Distributed Systems, FMOODS/FORTE'13, LNCS 7892. pp. 99–113. Springer (2013)
5. Lee, D., Netravali, A., Sabnani, K., Sugla, B., John, A.: Passive testing and applications to network management. In: 5th IEEE Int. Conf. on Network Protocols, ICNP'97. pp. 113–122. IEEE Computer Society (1997)
6. Núñez, M.: Formal testing of timed and probabilistic systems. In: 23rd Int. Conf. on Testing Software and Systems, ICTSS'11, LNCS 7019. pp. 9–14. Springer (2011)

Demostración de NDT-Driver: una herramienta de soporte a los mecanismos de transformación de NDT

J. A. García-García, M.J. Escalona

[1] Grupo de Investigación IWT2. Universidad de Sevilla, Sevilla, España.
julian.garcia@iwt2.org; mjescalona@us.es

Abstract. En el contexto del paradigma de ingeniería guiada por modelos (MDE), el uso de metodologías –como es el caso de NDT (Navigational Development Techniques) – ayuda a asegurar la calidad de los resultados durante el desarrollo software. Sin embargo, en el día a día de las empresas que trabajan bajo el paraguas de este tipo de metodologías, muy a menudo acaecen problemas que no deberían darse y que provocan en última instancia, que la aplicación de las fases metodológicas sean consideradas como una mera formalidad sin utilidad aparente. Además, la definición teórica de metodologías hace impracticable su aplicación en contextos empresariales debido al uso de una terminología demasiado abstracta (metamodelos, transformaciones, conceptos, etc.). Por esto, se hace necesario desarrollar herramientas de soporte metodológico que oculten esta terminología teórica para mejorar su aplicabilidad. Este artículo presenta una demostración de NDT-Driver, una herramienta de soporte de la metodología NDT que permite a las empresas aprovechar todo el potencial de MDE de una manera transparente, sin tener que conocer conceptos tan abstractos como el de transformación de modelos. Actualmente, esta herramienta está siendo utilizada de forma satisfactoria en entornos reales de diferentes ámbitos de negocio.

Keywords: Model-Driven Engineering, Herramienta Basadas en Modelos, Herramienta, NDT

1 Introducción, contexto y necesidad

El paradigma de ingeniería guiada por modelos [1] (MDE, Model Driven Engineering) surgió con el fin de hacer frente a la complejidad de las plataformas y la incapacidad de los lenguajes de tercera generación para el alivio de esta complejidad. Todo ello con el propósito de sistematizar lo máximo posible el ciclo de vida de desarrollo software. Para alcanzar este objetivo, MDE se basa, como forma primaria de expresión, en la definición de modelos (de diferente nivel de abstracción) y reglas de derivación entre ellos para generar modelos cada vez más específicos y cercanos a la plataforma final. Además, si se desarrolla herramientas eficientes para aplicar este paradigma, el proceso puede ser automatizable en gran parte.

Sin embargo, MDE no es fácil de aplicar en entornos reales debido a que su terminología no resulta atractiva ni comúnmente entendible para los equipos de desarrollo. Conceptos tales como modelos, metamodelos, reglas de derivación o

transformación, entre otros, no son notaciones comunes en el entorno empresarial y se consideran demasiado abstractos y complejos.

Este artículo presenta cómo la metodología NDT [2] (Navigational Development Techniques) aborda esta problemática con su herramienta de soporte NDT-Driver con la que es posible dar respuesta a las dificultades acaecidas durante la especificación de sistemas de información.

2 Visión general de la metodología NDT

Actualmente, NDT proporciona soporte a todas las fases del ciclo de vida software (Estudio de Viabilidad, Ingeniería de Requisitos, Análisis, Diseño, Implementación, Pruebas, y Mantenimiento). Este soporte pasa por la definición formal de un conjunto de metamodelos para definir cada una de sus fases y la definición de reglas de derivación con las que es posible generar modelos de una fase de manera sistemática a partir de otros modelos de otras fases. Por una parte, los metamodelos se formalizan mediante perfiles UML, con el aporte de OCL [5] para la definición de restricciones, y el soporte de la herramienta Enterprise Architect [4] como herramienta de modelado UML. Por otra parte, las reglas de derivación son descritas en lenguaje QVT [3].

Sin embargo, como hemos adelantado anteriormente, esta definición teórica de NDT se antoja impracticable en contextos empresariales debido al uso de una terminología demasiado abstracta. Se hizo, por tanto, necesario desarrollar herramientas que ocultasen esta terminología teórica para mejorar la aplicabilidad de NDT en entornos reales. Esto se logró con NDT-Suite [7], un paquete de herramientas que proporciona soporte al uso de la metodología NDT.

El abanico de herramientas que componen NDT-Suite es muy amplio como para realizar una demostración de forma completa. Por este motivo, este artículo se centra en una de ellas: NDT-Driver. Esta herramienta se describe en la siguiente sección.

3 NDT-Driver

NDT-Driver implementa en Java un conjunto de procedimientos automáticos para ejecutar cada una de las reglas de transformación QVT que define NDT. En este sentido, NDT-Driver es capaz de generar los modelos de la fase de Análisis desde el modelo de la fase de Requisitos, los modelos de la fase de Diseño desde los modelos de Análisis, y el modelo de pruebas de sistema de la fase de Pruebas desde el modelo de Requisitos. Además, la herramienta permite obtener el modelo Requisitos a partir de los requisitos capturados durante la fase de Estudio de Viabilidad del proyecto.

La Figura 1a muestra de la interfaz principal de NDT-Driver, desde la que es posible seleccionar qué transformación llevar a cabo, así como el modo de transformación (mencionado más adelante) deseado.

Asimismo, para facilitar el trabajo en proyectos que siguen ciclos de vida iterativos e incrementales, NDT-Driver permite generar cada modelo del mismo modo, es decir, siguiendo una estrategia incremental. Además, la herramienta permite seleccionar, para cada transformación, el modelo concreto que se desea generar. Por

ejemplo, la Figura 1b muestra la ventana para configurar la transformación de la fase de Requisitos a la fase de Análisis.

(a) Interfaz de Usuario Principal. **(b)** Ventana de Configuración.

Fig. 1. Herramienta NDT-Driver.

La aplicación automática de las reglas de transformación proporciona, sin duda, una posición aventajada al analista a la hora de especificar los distintos modelos de cada fase del ciclo de vida. Sin embargo, estos modelos generados de forma automática deben ser mejorados para completar y enriquecerlos en aras de obtener una versión definitiva.

Durante este proceso de mejora es posible que se detecten necesidades del cliente que no han sido considerados inicialmente, lo que puede conllevar la modificación del modelo origen para volver a generar el modelo en cuestión. Esto puede conllevar la pérdida de todo el trabajo realizado por el analista durante el proceso de mejora.

Como solución, NDT-Driver permite dos modos de transformación: *Reconstrucción* o *Actualización*. El primer modo consiste en volver a generar un modelo desde cero partiendo del modelo origen y descartando el modelo generado, si éste ha sido previamente generado. El segundo modo consiste en transformar sólo aquellos elementos del modelo original que han sido modificados.

Todas las características hacen de NDT-Driver, una herramienta que permite reducir cuantitativamente el tiempo dedicado a la especificación de cualquier proyecto que siga las directrices de la metodología NDT.

En la web del grupo de investigación IWT2[1] (sección "Aprendiendo NDT") el lector puede encontrar diferentes videos de demostración que muestran cómo se utiliza NDT-Driver. Asimismo, el lector puede encontrar en dicha web, un proyecto de prueba (proyecto del Hotel Ambassador) sobre el que ejecutar la herramienta.

4 Conclusiones y lecciones aprendidas

Este artículo presenta NDT-Driver: una herramienta sencilla e intuitiva para que cualquier empresa que desarrolla sus proyectos en el marco de la metodología NDT

[1] http://www.iwt2.org/

pueda aprovechar todo el potencial del paradigma MDE, sin tener que conocerlo teóricamente. De manera concreta, NDT-Driver automatiza todas las reglas de transformación QVT que define NDT

Como lección aprendida de nuestra experiencia [8; 9] en el uso de MDE, podemos concluir que su uso en contextos reales puede mejorar cuantitativamente los resultados del proyecto. De hecho, el uso de perfiles UML, y herramientas basadas en UML ofrecen una interfaz efectiva y atractiva para que cualquier tipo de usuario pueda trabajar de forma eficaz bajo el paraguas de una metodología enmarcada dentro del paradigma MDE (como es el caso de NDT).

Agradecimientos

Este artículo ha sido financiado por el proyecto MeGUS (TIN2013-46928-C3-3-R) del Ministerio de Ciencia e Innovación y el proyecto NDTQ-Framework (TIC-5789) de la Junta de Andalucía (España). Además, también ha sido financiado por la fundación Universia gracias a su programa de becas para estudiantes de doctorado con discapacidad.

Referencias

1. Schmidt, D. C. *Model-Driven Engineering*. IEEE Computer, Computer Society, vol. 39, no. 2, pp. 25-31, 2006.
2. Escalona MJ, Aragón G. *NDT. A model-driven approach for web requirements. IEEE Transactions on software engineering*, vol: 34, pp. 377-394. 2008.
3. OMG. *Documents Associated with Meta Object Facility (MOF) 2.0 Query/View/Transformation*. Object Management Group. URL: http://www.omg.org/spec/QVT/1.0/. 2008.
4. SparxSystems. *Enterprise Architect*. Website: www.sparxsystems.com.au, último acceso abril 2014.
5. ISO/IEC. *ISO/IEC 19507:2012 Information technology - Object Management Group Object Constraint Language (OCL)*. International Organization for Standardization, formal/2012-05-09, 2012.
6. García-García, J.A., Cutilla, C.R., Escalona, M.J., Alba, M., Torres, J. *NDT-Driver, a Java Tool to Support QVT Transformations for NDT*. In the 20th International Conference on Information Systems Development (ISD), DOI 10.1007/978-1-4614-4951-5_8, pp. 170-176., 2012.
7. García-García, J. A., Escalona, M., Domínguez-Mayo, F., Salido, A. *NDT-Suite: A Methodological Tool Solution in the Model-Driven Engineering Paradigm*. Journal of Software Engineering and Applications, 7, 206-217. doi: 10.4236/jsea.2014.74022. 2014.
8. Escalona, M. J., Garcia-Garcia, J. A., Mas, F., Oliva, M., and Del Valle, C. (2013) Applying model-driven paradigm: CALIPSOneo experience. Proceedings of the Industrial Track of the Conference on Advanced Information Systems Engineering 2013 (CAiSE'13), vol. 1017, pp. 25-32. Valencia, Spain.
9. Cutilla, C. R., García-García, J. A., Alba, M., Escalona, M.J., Ponce, J., and Rodríguez, L. (2011) Aplicación del paradigma MDE para la generación de pruebas funcionales; Experiencia dentro del proyecto AQUA-WS. Presented at the 6ª Conferência Ibérica de Sistemas e Tecnologias de Informação. ISBN: 978-989-96247-4-0.

DB-Main/Models: Un caso de estudio sobre la interoperabilidad de herramientas basada en MDE

Francisco Javier Bermúdez Ruiz[1], Jesús García Molina[1] y Oscar Díaz García[2]

[1] Departamento de Informática y Sistemas
Universidad de Murcia
{fjavier,jmolina}@um.es
[2] Departamento de Lenguajes y Sistemas Informáticos
Universidad del Pais Vasco
{oscar.diaz}@ehu.es

Resumen. La Ingeniería del Software Dirigida por Modelos (MDE) promueve la utilización sistemática de modelos para mejorar el nivel de abstracción en la construcción de software y con ello proporcionar mejoras significativas en la productividad. Las técnicas MDE además de ser útiles para automatizar cualquier etapa del ciclo de vida del desarrollo de software, también facilitan la integración de espacios tecnológicos y la interoperabilidad de herramientas y sistemas. En este artículo se presenta un caso de estudio sobre la integración de la herramienta DB-Main en el espacio tecnológico MDE (Modelware) y la interoperabilidad con otras herramientas haciendo uso de modelos. DB-Main es una herramienta bien conocida en Ingeniería de Datos que ha sido desarrollada por la empresa ReveR, spin-off surgida del grupo de investigación Precise (Universidad de Namur, Bélgica). DB-Main ofrece funcionalidad relacionada con procesos de diseño, transformaciones de datos, ingeniería inversa de datos, análisis de programas y evolución de datos. El trabajo muestra un análisis de varias posibles estrategias de interoperabilidad basada en modelos aplicadas a DB-Main, señalando algunas de las ventajas e inconvenientes encontrados y discutiendo la conveniencia de cada una de las alternativas presentadas.

Palabras clave: Ingeniería del Software Basada en Modelos, Ingeniería de datos, DB-Main, Integración, Interoperabilidad

1 Introducción

DB-Main[3] es una herramienta CASE que asiste a desarrolladores y analistas en diversas tareas dentro de un proceso de ingeniería de datos, como son el diseño de base de datos, ingeniería inversa, reingeniería, mantenimiento y evolución de datos. Surgió en 1993 en el seno del grupo de investigación *Precise* (Universidad de Namur) y actualmente es soportada por la empresa *ReveR*, una spin-off

[3] http://www.db-main.be

surgida en dicho grupo. *DB-Main* es una herramienta gratuita, madura y ampliamente extendida en la comunidad de Ingeniería de Datos. Permite representar sistemas de información a distintos niveles y para varios paradigmas de representación de datos, como el relacional y el orientado a objetos.

El trabajo que se presenta ha sido realizado dentro de la estancia predoctoral que el primer autor realiza en el grupo *Precise* y cuyo objetivo es estudiar la aplicación de MDE en los procesos de ingeniería inversa de datos. En particular, este artículo se centra en describir y analizar las estrategias basadas en MDE para la integración de *DB-Main* con otros sistemas.

En [11] se define la *interoperabilidad* como *"la habilidad de dos o más sistemas o componentes para intercambiar información y usar la información intercambiada"*. La interoperabilidad de sistemas trata con aspectos sintácticos (formatos que usan los sistemas para almacenar información) y semánticos (cómo los sistemas interpretan información), como se indica en [2]. Por tanto la interoperabilidad de sistemas debe ser tratada a nivel de datos y a nivel operacional.

Una de las aplicaciones de MDE es el uso de modelos y transformaciones de modelos para lograr la interoperabilidad entre sistemas (herramientas, tecnologías, procesos de negocio, etc.) a un alto nivel de abstracción. *Model-driven Interoperability* (*MDI*) es el área que se ocupa de esta aplicación de MDE y nuestro trabajo se enmarca en una de las dimensiones del área MDI: la interoperabilidad de herramientas basada en modelos [3] [9].

Con el fin de facilitar su uso desde aplicaciones y herramientas, *DB-Main* ofrece un API de acceso, así como dos formatos de representación de los datos manejados en un proyecto: un esquema XML y un fichero con un formato propietario llamado LUN. En este artículo veremos cómo MDE puede facilitar y extender la interoperabilidad de *DB-Main* con otras herramientas. Se ha desarrollado como un plugin de *DB-Main* llamado *DB-Main/Models* que proporciona un puente (*bridge*) bidireccional entre *DB-Main* y la tecnología MDE (también conocida como Modelware), el cual está formado por un metamodelo que representa la información manejada por la herramienta junto con un *inyector* que genera modelos a partir de dicha información y un *extractor* que realiza la operación inversa. La existencia de tres diferentes formas de interacción con *DB-Main* ofrece diferentes alternativas de implementación para *DB-Main/Models* que son analizadas. También se discute como este *bridge* podría ser usado para conectar *DB-Main* con otras herramientas por medio de soluciones MDE.

Por tanto la contribución de este artículo es doble. Por un lado se presenta un caso de estudio de interoperabilidad de herramientas basado en MDE y por otro lado se ofrece un análisis de cada una de las opciones de implementación de dicha integración.

El artículo se organiza como se describe a continuación. En la Sección 2 se muestra el trabajo relacionado. En la Sección 3 se presenta la descripción del problema introduciendo el esquema básico de interoperabilidad basada en modelos. En la Sección 4 se detallan diferentes estrategias para la implementación de *DB-Main/Models*. En la Sección 5 se presentan los diferentes casos de integración de *DB-Main* con otros sistemas haciendo uso de *DB-Main/Models*. Por último,

en la Sección 7 se presentan las conclusiones sobre las alternativas identificadas para implementar la propuesta.

2 Trabajo relacionado

La integración de herramientas ha suscitado gran interés desde los inicios de la ingeniería del software. En [14] se analiza la integración de herramientas desde varias dimensiones y una reciente revisión de la literatura con una agenda de investigación futura puede encontrarse en [15]. De acuerdo al marco definido en [14], nuestro trabajo se centra en el problema de la interoperabilidad de herramientas, esto es, en cómo facilitar la integración de herramientas que manejan formatos de datos diferentes. En la revisión de literatura mostrada en [15] no se consideran los trabajos de integración basada en modelos que entonces empezaban a emerger, sino que sólo aparece como novedoso el uso de XMI como formato de intercambio de datos.

El concepto de *espacio tecnológico* fue introducido en [12], como forma de definir tecnologías a un alto nivel de abstracción. Esta noción fue utilizada en [4] para establecer puentes entre diferentes tecnologías (por ejemplo, Grammarware, Ontologías y Modelware). Los autores definen para cada espacio tecnológico una arquitectura de varias capas similar a la arquitectura de metamodelado usada en MDE con el fin de establecer un mapping entre los artefactos de dos tecnologías concretas. Esta idea ha sido llevada a la práctica en la construcción de varios lenguajes de transformación que permitían inyectar en modelos conformes a un metamodelo los artefactos manejados en una tecnología: Gra2MoL [5], API2MoL [6] y Schemol [7]. Gra2MoL permite establecer un *mapping* entre una gramática y un metamodelo para inyectar código conforme a la gramática en modelos; APi2MoL permite obtener un metamodelo a partir de un API Java, junto con un inyector de objetos del API a modelos y un extractor que crea objetos del API a partir de modelos; Schemol permite establecer un mapping entre un esquema relacional y un metamodelo para inyectar datos de tablas en modelos. En la construcción de *DB-Main/Models* se han utilizado los dos primeros, Gra2MoL para inyectar ficheros LUN y API2MoL para inyectar los objetos del API JIDBM.

La utilización de MDE para la integración de herramientas fue planteada en [9]. En este trabajo se proponía usar *AWM* (Atlas Weaver Model)[4] para definir un modelo de correspondencias (weaving model) que expresase el mapping semántico entre los metamodelos de las dos herramientas a ser integradas. El enfoque fue ilustrado con dos herramientas de bug tracking. En [3] se presenta una solución más general para el caso en el que los metamodelos de las herramientas no hayan sido creados con el mismo lenguaje de metamodelado. El enfoque genera de forma semi-automática las transformaciones del *bridge* entre herramientas y se aplica a la creación de dos *bridges* para EMF, uno con el metamodelo de DSL Tools[5] y otro con SQL Server Modeling[6]. El *bridge* pre-

[4] http://www.eclipse.org/gmt/amw
[5] http://msdn.microsoft.com/en-us/library/bb126327.aspx
[6] http://www.microsoft.com/en-us/download/details.aspx?id=24113

sentado en este artículo para DB-Main supone que los modelos de herramientas son creados con el mismo meta-metamodelo (Ecore/EMF) y la transformación para los metamodelos es creada manualmente. Esta solución está orientada al caso muy común de herramientas no MDE o herramientas con metamodelos Ecore/Eclipse.

Realmente, la integración de herramientas basada en MDE es parte de un área más grande: *Model-driven Interoperability* (*MDI*), que aborda el estudio y aplicación de MDE en la interoperabilidad de las empresas y sus sistemas. Un marco conceptual para MDI es descrito en [8] y una plataforma concreta en [13].

3 Descripción del problema

Como hemos indicado, *DB-Main* es una herramienta madura que ofrece una rica funcionalidad para tareas de Ingeniería de Datos. Para su utilización desde otros sistemas, ya sean aplicaciones CASE u otras herramientas, ofrece tres alternativas: el *API JIDBM* y dos tipos de representación de datos como son un *esquema XML* y el formato propietario *LUN*. JIDBM es un API Java que ofrece un conjunto de métodos para manipular los datos y metadatos de un proyecto *DB-Main*; el documento XML se genera mediante una opción de exportación de datos de históricos de esquemas para una base de datos en *DB-Main*, una característica que está disponible en una versión beta, aunque no incorporada a la última versión de la herramienta a la que hemos tenido acceso gracias a la colaboración con en el grupo *Precise*; y el fichero LUN usa un formato propietario para representar toda la información de un proyecto *DB-Main*.

DB-Main podría ser integrado con otras herramientas con el fin de aprovechar funcionalidades que no soporta, lo cual requeriría una conversión del formato *DB-Main* al formato de importación de esas herramientas o bien acceder mediante el API JIDBM a los datos del repositorio *DB-Main* y representarlos en el formato destino (considerando que la herramienta destino dispone de importación de datos). Por ejemplo, podría ser integrado con herramientas de modelado UML como *MagicDraw* de *NoMagic*[7] o herramientas CASE como *Enterprise Architect* de *Sparx*[8] que pueden importar modelos Ecore y modelos XMI, respectivamente, y que disponen de facilidades para la generación de código. Por otro lado, una aplicación podría acceder a *DB-Main* para utilizar alguna funcionalidad como la realización del análisis de un esquema de datos. En este caso, sería preciso transformar el esquema de datos desde el formato origen manejado por la aplicación al formato manejado en *DB-Main* (en la actualidad, *DB-Main* no soporta la importación de archivos XML) o bien usar el API bidireccional JIDBM. La Fig. 1 ilustra las dos direcciones de la posible integración de *DB-Main* con otras herramientas.

Siempre que se decida conectar *DB-Main* con otro sistema o usarlo desde una aplicación que maneja datos en otro formato diferente al definido en *DB-Main*, en vez de crear desde cero una integración ad-hoc, es posible hacer uso de

[7] http://www.nomagic.com
[8] http://www.sparxsystems.com

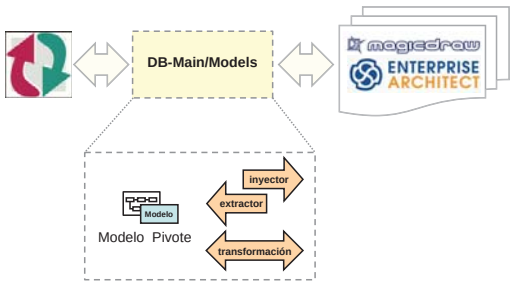


Fig. 1. Integración basada en modelos de *DB-Main* con otras herramientas

MDE para facilitar la interoperabilidad y automatizar la creación del software necesario para disponer de la integración en los dos sentidos.

La interoperabilidad de dos herramientas requiere establecer una correspondencia entre los formatos utilizados por cada una de ellas para representar la información que manejan. La utilización de MDE para implementar la interoperabilidad se basa en la definición de un metamodelo pivote que represente la información manejada por cada herramienta [2]. La Fig. 2 muestra el esquema de interoperabilidad basada en modelos para dos herramientas A y B, cada una con un formato propietario para representar la información. Los metamodelos A y B son los metamodelos pivote y ambos comparten el mismo meta-metamodelo (por ejemplo, Ecore). Una transformación bidireccional entre ambos metamodelos se encarga de establecer el *mapping* semántico entre las dos herramientas.

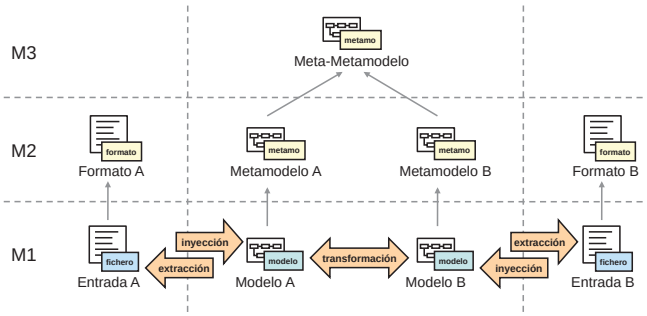


Fig. 2. Esquema de interoperabilidad basada en modelos (tomado de [2])

Dado que cada herramienta utiliza su propio formato para representar información, el *mapping* semántico requiere de un *mapping* sintáctico que establezca una transformación bidireccional entre el formato de una herramienta y su correspondiente metamodelo. Esta transformación se implementa por medio de un inyector que genera un modelo a partir de información expresada en el formato propietario y de un extractor que realiza la operación inversa, como mues-

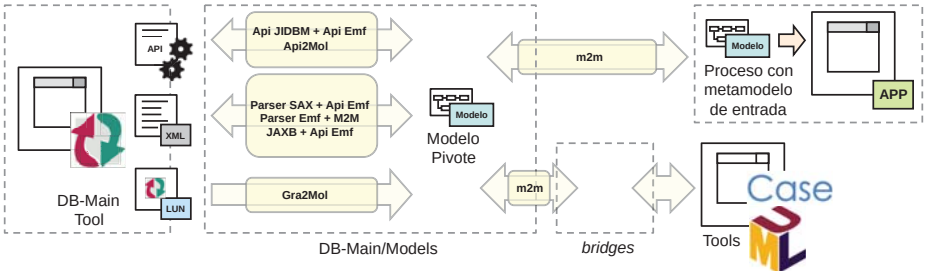


Fig. 3. Arquitectura de la solución DB-Main/Models

tra la Fig. 2. Este *mapping* sintáctico establece un *bridge* de integración de una herramienta en la tecnología MDE (Modelware). En este trabajo se presenta un puente de esta naturaleza para *DB-Main* al cual hemos llamado *DB-Main/Models* y que consta de:

1. El *metamodelo pivote DB-Main* que representa los datos o artefactos manejados por la herramienta.
2. Los *inyectores* que convierten datos (XML, LUN y API JIDBM) del formato *DB-Main* a modelos conforme al metamodelo *DB-Main*.
3. Los *extractores* que generan datos *DB-Main* a partir de modelos *DB-Main*.

Dada una herramienta, la existencia de un *bridge* a MDE facilita su integración con otras herramientas puesto que se pueden reutilizar inyectores y extractores de modo que el problema se reduce a crear un *bridge* para la otra herramienta, si no está basada en modelos. En la próxima sección discutiremos las diferentes estrategias para implementar el inyector y extractor de *DB-Main/Models*, y en la Sección 5 se analizará como el puente puede ser usado para integrar *DB-Main* con otras herramientas. La existencia de varias alternativas para implementar *DB-Main/Models* las variantes que ofrece *DB-Main* para acceder a su información: API JIDBM, documento XML y fichero LUN

4 Estrategias para la implementación de *DB-Main/Models*

La Fig. 3 ilustra la arquitectura del *bridge* implementado para la integración de *DB-Main* en MDE. Se ha implementado como un plugin de la herramienta *DB-Main*, haciendo uso del framework proporcionado por Eclipse para aplicar MDE: Eclipse Modeling Framework (EMF)[9]. La solución consta de un metamodelo pivote junto a los inyectores y extractores. En la implementación se aplica un proceso iterativo de modo que el metamodelo pivote se va enriqueciendo en sucesivas iteraciones. Esto, como es lógico, supone que los inyectores y extractores se van completando también en cada iteración.

[9] http://www.eclipse.org/modeling/emf

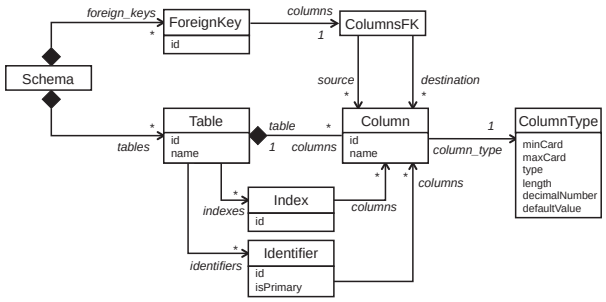


Fig. 4. Metamodelo pivote *DB-Main*

A continuación se describen los procesos de creación del metamodelo, inyectores y extractores que tendrán lugar en cada iteración.

Creación del metamodelo pivote

El *metamodelo pivote* es el elemento central de la arquitectura y representa la información manejada en *DB-Main*, como por ejemplo el esquema de datos actual, el historial de versiones de esquemas y los datos del proyecto. En una primera iteración de la implementación sólo se ha considerado información relacionada con el esquema de datos y en futuras iteraciones se considerará el resto (por ejemplo, la información sobre el histórico de versiones de un esquema es esencial para analizar la evolución de la base de datos). La Fig. 4 muestra el metamodelo *DB-Main* definido actualmente y que representa un esquema compuesto por un conjunto de tablas formadas a su vez por columnas con su tipo de datos, y además se representan identificadores, claves foráneas e índices. El metamodelo pivote ha sido diseñado de acuerdo a la estructura de datos usada en *DB-Main* con el objetivo de simplificar las tareas de inyección y extracción.

Creación de los inyectores

Una vez definida una primera versión del metamodelo, es posible crear los inyectores precisos para crear modelos *DB-Main* a partir de los artefactos de la herramienta. Los modelos pueden ser inyectados o bien usando el API JIDBM o los archivos manejados por la herramienta: ficheros de exportación en XML y ficheros de proyecto LUN. En la Fig. 5 se presentan cuatro esquemas que resumen las estrategias de inyección y extracción en *DB-Main/Models* de acuerdo a los artefactos de la herramienta.

En caso de usar *JIDBM*, se han identificado dos posibles estrategias para la inyección, tal y como se muestra en la Fig. 5(a):

1. *Uso del API de persistencia de EMF.* Se implementa un programa Java que usa JIDBM para acceder a los datos de *DB-Main* (no es necesaria tener en ejecución la herramienta) y el API EMF para crear un modelo pivote *DB-Main* que representa la información recuperada sobre el esquema de datos.

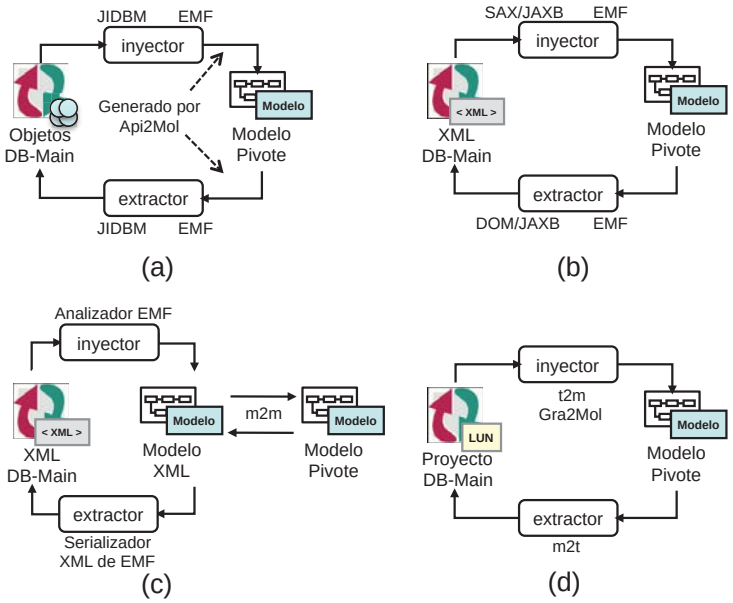


Fig. 5. Inyectores y extractores en *DB-Main/Models*

Se requiere por tanto de la implementación manual del inyector usando el API de persistencia en modelos de EMF.

2. *Uso de API2MoL.* Dada una API Java, API2MoL [6] crea automáticamente el metamodelo del API así como el correspondiente inyector y extractor. La entrada a API2MoL debe ser un programa Java que utilice el API JIDBM para recuperar los datos de un proyecto *DB-Main*. API2MoL nos evitaría implementar manualmente la tarea de deserializar (inyectar) la información en un modelo por medio del API EMF. Aunque automatizar esta tarea supone un ahorro en el esfuerzo de crear un inyector con respecto a la anterior estrategia, se hace necesario el aprendizaje y la práctica de API2MoL (una herramienta de investigación prototípica y poco documentada), lo cual puede hacer más rentable la implementación manual.

En caso de usar el *documento XML* para inyectar un modelo *DB-Main*, se han detectado tres estrategias, como se muestra en las Fig. 5(b) y 5(c):

1. *Uso del analizador SAX junto al API de persistencia EMF.* Se implementa un parser XML utilizando el analizador SAX en Java (JAXP[10]). La información extraída en el análisis del documento XML se convierte en un modelo *DB-Main* mediante el uso del API EMF. A pesar de que JAXP dispone de un analizador de datos DOM que carga en memoria de una sola vez toda la información del documento XML, las condiciones de consulta de los datos

[10] http://jcp.org/en/jsr/detail?id=63

XML favorecen el uso de SAX. El recorrido del documento XML es secuencial y tan solo se requiere una pasada por lo que cada nodo del árbol XML es consultado una sola vez. El análisis DOM requiere un mayor consumo de memoria y es aconsejado principalmente en escenarios donde los nodos se consultan de manera aleatoria (no secuencial) e incluso más de una vez.

2. *Uso del analizador/inyector genérico de EMF para XML junto a una transformación modelo-a-modelo.* Se utiliza el analizador e inyector de modelos genérico que ofrece EMF para inyectar documentos XML. EMF requiere el esquema XML del documento de entrada y genera automáticamente un metamodelo que representa los mismos elementos del esquema XML. El inyector convierte cualquier documento conforme al esquema XML en un modelo conforme al metamodelo creado. Sería necesaria entonces una transformación modelo-a-modelo (m2m) para convertir el modelo obtenido en un modelo pivote *DB-Main*. Esta transformación podría escribirse con un lenguaje de transformaciones m2m o creando una aplicación Java que use el API EMF para consultar el modelo de entrada y generar el de salida.
3. *Uso de JAXB junto al API de persistencia EMF.* A partir del esquema del documento XML exportado se genera automáticamente un modelo de objetos Java conforme al esquema XML utilizando el *mapper* XML-Objetos de Java (JAXB[11]). Es necesario utilizar el desempaquetador (*unmarshall*) de JAXB para cargar los datos XML en el modelo de objetos Java. Entonces se explora el modelo de objetos mediante una aplicación Java y se usa el API EMF para persistir la información obtenida en el modelo pivote *DB-Main*.

La tercera alternativa que ofrece *DB-Main* para la inyección de modelos son los archivos LUN (ver Fig. 5(d)), los cuales contienen la información sobre el esquema de datos, el histórico de versiones de esquemas y las transformaciones aplicadas dentro del proceso de ingeniería de datos. Hemos identificado dos posibles estrategias para inyectar estos archivos.

1. Escribir un analizador-inyector en Java que realice el análisis sintáctico de un archivo LUN y utilice el API EMF para crear el correspondiente modelo.
2. Dado que el contenido de los archivos LUN conforma a una gramática se podría usar el lenguaje Gra2MoL definido para inyectar modelos a partir de texto conforme a una gramática [5]. Gra2MoL permite escribir transformaciones texto-a-modelo que definen una correspondencia entre los elementos de una gramática y los de un metamodelo destino. La ejecución de una transformación Gra2MoL tiene como entrada un archivo cuyo texto es conforme a una gramática y como salida el modelo conforme al metamodelo destino. En nuestro caso, la transformación Gra2MoL establecerá una correspondencia entre la gramática del fichero LUN y el metamodelo pivote *DB-Main*.

La utilización de un lenguaje específico para inyectar modelos a partir de texto conforme a gramáticas como es Gra2MoL automatiza la construcción y el uso de los analizadores e inyectores necesarios, frente a la tarea manual de

[11] http://jcp.org/en/jsr/detail?id=222

programar un analizador en Java. No obstante, por la complejidad y variabilidad del contenido del fichero LUN, y puesto que mucha de la información contenida queda determinada por tipos de datos y posiciones relativas entre ellos, el uso de Gra2MoL en este escenario no es sencillo y la tarea de definir la transformación en Gra2MoL para el formato LUN es costosa.

Creación de los extractores

Los extractores realizan el proceso inverso a los inyectores, esto es, dado un modelo pivote *DB-Main* generan los artefactos de *DB-Main*: invocaciones al API JIDBM, documentos XML y archivos LUN.

En el caso de usar el *API JIDBM*, las dos estrategias mencionadas anteriormente para la inyección son bidireccionales dado que JIDBM también tiene métodos para la creación y actualización de la información manejada en *DB-Main*. Por tanto, se podría usar JIDBM junto al API EMF para implementar la generación, aunque el uso de API2MoL permite generar automáticamente el inyector y extractor, tal y como se ilustra en la Fig. 5(a).

En el caso de generar un *documento XML* conforme al esquema XML manejado por *DB-Main* podemos usar también las estrategias usadas para la inyección pero en sentido inverso:

1. EMF para recorrer el modelo DB-Main y en este caso el API DOM o el *mapper* JAXB para generar el documento XML (Fig. 5(b)). Se aconseja el uso de DOM pues para la creación de árboles XML esta tecnología permite mayor flexibilidad en el manejo del nuevo documento XML.
2. Transformación m2m para generar el modelo conforme al metamodelo XML correspondiente al esquema XML de *DB-Main* y autogenerado durante la inyección por EMF. Posteriormente se usa la herramienta disponible en EMF para serializar el modelo XML en un documento XML conforme al esquema XML que corresponde al metamodelo XML (Fig. 5(c)). No obstante la obtención del archivo XML no tendría utilidad actualmente puesto que *DB-Main* no soporta todavía la importación.

Cabe destacar que las estrategias que combinan el API EMF con DOM o JAXB pueden ser bidireccionales aunque completamente manuales mientras que la estrategia que usa el analizador/serializador de EMF sólo requiere dos transformaciones m2m, o incluso una única transformación entre el metamodelo pivote y el metamodelo que corresponde al esquema XML si se usara un lenguaje de transformaciones m2m que soporte bidireccionalidad, como es el caso de QVT[12].

Por último, en el caso de generar el archivo LUN a partir del modelo pivote *DB-Main* se podría hacer uso de una transformación modelo-a-texto (m2t) utilizando lenguajes como Acceleo[13] o Xpand[14] (Fig. 5(d)). Sin embargo, la complejidad de la gramática del formato LUN convierte su uso en la vía más costosa de implementar para obtener tanto el inyector como el extractor.

[12] http://www.omg.org/spec/QVT/1.1/
[13] http://www.eclipse.org/acceleo
[14] http://www.eclipse.org/modeling/m2t/?project=xpand

5 Casos de integración de *DB-Main*

Como se muestra en la Fig. 2 y señalamos en la Sección 3, dada una particular herramienta A, una vez se dispone para ella de un *bridge* bidireccional al Modelware formado por inyectores y extractores para el metamodelo pivote (*mapping* sintáctico), la interoperabilidad con otra herramienta B requiere que ella también disponga del metamodelo pivote de la herramienta de modo que se pueda implementar una transformación modelo-a-modelo bidireccional entre ambos metamodelos (*mapping* semántico). Esta transformación m2m sería única si el lenguaje soporta transformaciones bidireccionales (QVT sin duda sería una posible opción) o bien habría que implementar dos transformaciones, una para cada sentido, si se usan lenguajes m2m con transformaciones unidireccionales, como es el caso de lenguajes muy extendidos como ATL[15] y Epsilon ETL[16].

En cuanto a la integración de B en el Modelware podemos encontrar dos casos alternativos a la construcción del *bridge* formado por inyector y extractor. Una posibilidad es que sólo se requiera interoperabilidad en un sentido y en ese caso sólo sería necesario la construcción del inyector o extractor de B; por ejemplo, si A sólo requiriese cierta funcionalidad de B, sería suficiente con crear el extractor de B. Por otra parte, puede suceder que no sea necesario crear tal *bridge* para B si la herramienta B ya soportase la exportación/importación de modelos Ecore.

En el caso de nuestro *bridge DB-Main/Models*, la integración con otra herramienta requiere la transformación del modelo pivote *DB-Main* a un modelo de la otra herramienta o bien el proceso inverso, dependiendo de si *DB-Main* requiere funcionalidad de la otra herramienta o al revés. A continuación se describen las diferentes opciones de integración con otras herramientas:

1. *Integración con herramientas que importan/exportan modelos Ecore.* En este caso sólo sería necesario implementar dos transformaciones m2m entre el modelo pivote de *DB-Main* y los modelos de entrada de la herramienta destino (o sólo una transformación si es bidireccional). Si el salto semántico entre los dos metamodelos pivote fuera alto se podría introducir alguna etapa intermedia que generase modelos intermedios y así simplificar la implementación del *mapping* semántico. Un ejemplo de este tipo de integración sería la integración de *DB-Main* con *MagicDraw* para, por ejemplo, visualizar un esquema de datos como un modelo UML o generar código. Otro ejemplo sería la utilización de *DB-Main* desde una herramienta MDE, como el framework de modernización de datos descrito en [1]. En este caso sería necesario implementar las transformaciones entre los modelos manejados en esta solución y el modelo pivote.
2. *Integración con herramientas que manejan datos en formato propietario y/o acceso basado en una API.* En este caso es necesario crear un *bridge* para integrar la herramienta en el Modelware e implementar la transformación entre el metamodelo *DB-Main* y el metamodelo pivote definido para la herramienta. Un ejemplo sería la herramienta CASE de base de datos *Enterprise*

[15] https://www.eclipse.org/atl
[16] http://www.eclipse.org/epsilon/doc/etl

Architect que ofrece tanto un fichero de importación/exportación de datos en XMI como un API Java para el acceso a su repositorio de modelos (con funciones de consulta y actualización). De nuevo la necesidad de establecer una cadena de transformaciones para la transformación entre metamodelos pivotes dependerá del salto semántico entre ambas representaciones.

6 Discusión

Como se ha visto en la Sección 5, dado que *DB-Main* presenta tres posibles vías de acceso a sus datos, existe una amplia variedad de estrategias para la implementación del *bridge*. La elección entre una u otra vendrá dada principalmente por el grado de automatización (con la consiguiente reducción de esfuerzo de desarrollo), las capacidades relativas a la bidireccionalidad en la interacción y el esfuerzo en la utilización de la tecnología de implementación. La siguiente tabla resume el comportamiento de cada enfoque según los aspectos comentados.

Tabla 1. Comparación de estrategias

	Estrategia	Automática	Bidireccional	Sencilla
API JIDBM	API Emf	✗	✔	✔
	API2MoL	✔	✔	✗
Fichero XML	SAX/DOM + Emf	✗	✔	✔
	Emf XML + m2m	✓	✔	✔
	JAXB + Emf	✗	✔	✔
Proyecto LUN	Gra2MoL	✔	✗	✗
	Analiz./Inyect. Java + Emf	✗	✗	✗

API2MoL se presenta como mejor opción puesto que genera automáticamente el metamodelo del API así como el inyector y el extractor, además de ofrecer bidireccionalidad en la solución. Si el metamodelo generado fuese apropiado como metamodelo pivote, solo sería necesario disponer de un programa que, usando el API JIDBM, construyese los objetos de datos a inyectar en el modelo pivote, obteniendo así una alto grado de automatización, de acuerdo a [5]. Si no fuera apropiado, sería necesario implementar una transformación de modelos desde el modelo inyectado al modelo pivote. Sin embargo hay que considerar que se trata de una herramienta académica, descontinuada y que carece de suficiente documentación para su aprendizaje.

Desde el punto de vista técnico hemos identificado algunas limitaciones de APi2MoL en el caso de *DB-Main/Models*. Por un lado, es preciso tener en memoria los objetos de datos a ser inyectados, ya que el proceso arranca accediendo al objeto raíz de la estructura de objetos del API (el objeto `Schema` en el API JIDBM). Además hemos encontrado dificultades para establecer relaciones de orden entre las llamadas al API y tampoco se soporta el uso de patrones de creación [10] para crear las estructuras de datos.

Tanto la opción de usar el analizador/inyector XML como la del serializador de EMF proporcionan cierta automatización (en la tabla se indica mediante una marca de menor tamaño) y sólo es necesario generar las transformaciones m2m entre el modelo generado automáticamente que se corresponde con el esquema XML, y el metamodelo pivote. Esta solución se considera parcialmente bidireccional puesto que aunque la transformación m2m puede ser bidireccional y el analizador/serializador EMF pueden ser usados directamente, los desarrollos actuales de la herramienta *DB-Main* no contemplan la importación de archivos XML (solo exportación). Comparado con el uso de API2MoL, esta segunda estrategia supone un coste adicional de escribir una o dos transformaciones m2m, dependiendo de si el lenguaje de transformaciones soporta o no bidireccionalidad. Por otro lado, el esfuerzo requerido para la implementación de estas estrategias es considerablemente menor al uso de API2MoL.

Por último, el uso del lenguaje Gra2MoL para la inyección de ficheros LUN y de una transformación m2t para generar un fichero LUN a partir de un modelo pivote supone, sobre el papel, una buena alternativa con un alto grado de automatización, aunque no ofrece bidireccionalidad. Sin embargo, la complejidad del formato LUN (en términos de gramática y cantidad de información contenida) encarecen en gran medida el esfuerzo para implementar esta estrategia.

7 Conclusiones

En este trabajo se ha presentado un análisis de las diferentes alternativas implementadas para la construcción de un *bridge* que soporte la integración de la herramienta *DB-Main* en el Modelware. El *bridge* está formado por el metamodelo pivote de la herramienta junto con el inyector y extractor. Dado que *DB-Main* ofrece tres diferentes vías para la interacción (API, XML y formato propietario), el análisis de su integración con otros sistemas nos ha permitido mostrar la variedad de técnicas que ofrece MDE para facilitar la implementación de un soporte a la interoperabilidad de sistemas: un lenguaje específico para la inyección (Gra2MoL), una herramienta que automatiza la creación de bridges en torno a una tecnología basada en un API Java (API2MoL), inyectores y extractores (serializadores) de XML, y transformaciones de modelos. La creación del modelo pivote que representa la información del sistema a ser integrado es una técnica clave que permite la reutilización de inyectores y extractores.

Finalmente cabe destacar que las estrategias discutidas en este artículo para la construcción de un *bridge* para la herramienta *DB-Main* son aplicables a otras herramientas, eligiendo una u otra en función de la disponibilidad o no de un API y de ficheros de datos (intercambio XML o formato propietario).

Como trabajo futuro se contempla la integración de *DB-Main* con otras herramientas (e.g. *MagicDraw*) implementando su correspondiente *bridge* (e.g. *MagicDraw/Models*) y definiendo el mapping semántico entre los metamodelos pivote. Además, es necesario analizar e implementar nuevas vías de integración en un *bridge* considerando otras métodos de acceso a los datos de una herramienta (e.g. acceso directo a la base de datos de una herramienta, si es posible).

Agradecimientos

Este trabajo es parte del Proyecto GUIZMO y ha sido financiado con la ayuda 15389/PI/10 de la Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia.

Referencias

1. Bermúdez Ruiz, F.J., García Molina, J.: Un framework basado en modelos para la modernización de datos. In: Actas de las XII JISBD. pp. 477–490 (2012)
2. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice. Synthesis Lectures on Soft. Eng., Morgan & Claypool Publishers (2012)
3. Bruneliere, H., et al.: Towards model driven tool interoperability: Bridging eclipse and microsoft modeling tools. In: Kühne, T., Selic, B., Gervais, M.P., Terrier, F. (eds.) EC-MFA. LNCS, vol. 6138, pp. 32–47. Springer (2010)
4. Bézivin, J., Kurtev, I.: Model-based technology integration with the technical space concept. In: Procs. of the Metainformatics Symposium. Springer-Verlag (2005)
5. Canovas Izquierdo, J.L., Garcia Molina, J.: Extracting Models from Source Code in Software Modernization. Software & Systems Modeling (2012)
6. Cánovas Izquierdo, J.L., Jouault, F., Cabot, J., García Molina, J.: Api2mol: Automating the building of bridges between apis and model-driven engineering. Information and Software Technology 54(3), 257–273 (2012)
7. Díaz, O., Puente, G., Cánovas Izquierdo, J.L., García Molina, J.: Harvesting models from web 2.0 databases. Software and System Modeling 12(1), 15–34 (2013)
8. Elvesæter, B., et al.: Towards an interoperability framework for model-driven development of software systems. In: Konstantas, D., et al. (eds.) Interoperability of Enterprise Software and Applications. pp. 409–420. Springer London (2006)
9. Fabro, M.D.D., Bézivin, J., Valduriez, P.: Model-driven tool interoperability: An application in bug tracking. In: Meersman, R., Tari, Z. (eds.) OTM Conferences (1). LNCS, vol. 4275, pp. 863–881. Springer (2006)
10. Gamma, E., et al.: Design Patterns. Addison-Wesley, Boston, MA (1995)
11. Geraci, A.: IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries. IEEE Press, Piscataway, NJ, USA (1991)
12. Kurtev, I., Bézivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: CoopIS, DOA'2002 Federated Conferences, Industrial track (2002)
13. Milanovic, N., et al.: Model-based interoperability of heterogeneous information systems: An industrial case study. In: Paige, R.F., et al. (eds.) ECMDA-FA. LNCS, vol. 5562, pp. 325–336. Springer (2009)
14. Thomas, I., Nejmeh, B.A.: Definitions of tool integration for environments. IEEE Software 9(2), 29–35 (1992)
15. Wicks, M.N., Dewar, R.G.: A new research agenda for tool integration. Journal of Systems and Software 80(9), 1569–1585 (2007)

Automatizando el desarrollo de editores gráficos cognitivamente eficaces

David Granada, Ángel Moreno, Juan M. Vara, Verónica A. Bollati, Esperanza Marcos

Grupo de Investigación Kybele, Universidad Rey Juan Carlos,
Calle Tulipán S/N, 28933 Móstoles, Madrid, Spain
{david.granada,angel.moreno,juanmanuel.vara,veronica.bollati,esperanza.marcos}@urjc.es
http://www.kybele.es

Resumen. Los lenguajes específicos de dominio juegan un papel fundamental en el desarrollo de software dirigido por modelos, en cuanto permiten obtener una mayor expresividad y facilidad de uso respecto a un lenguaje de propósito general. La creación de editores gráficos para trabajar con este tipo de lenguajes no es una tarea trivial, aunque actualmente existen diferentes entornos de desarrollo que proporcionan el soporte para llevar a cabo dicha creación. Mediante el análisis de las principales características de dichos entornos, hemos identificado algunos aspectos de mejora relacionados con la eficacia cognitiva de las notaciones visuales y con la automatización de todo el proceso de desarrollo de un editor gráfico. Por este motivo, en este trabajo introducimos `CEViNEdit`, una herramienta basada en GMF que proporciona mecanismos que permiten guiar la selección de las variables visuales que componen la notación, evaluar la eficacia cognitiva de dicha selección y automatizar la generación del editor gráfico.

Palabras clave: Ingeniería Dirigida por Modelos, Lenguaje Específico de Dominio, Notación Visual, Automatización, Eficacia Cognitiva.

1. Introducción

Los Lenguajes Específicos de Dominio (*Domain Specific Languages*, DSL) juegan un papel clave en prácticamente cualquier propuesta que aplique los principios de la Ingeniería Dirigida por Modelos (*Model Driven Engineering*, MDE) [1]. El hecho de que están construidos con el objetivo de diseñar soluciones para un dominio concreto redunda en mayor expresividad y facilidad de uso que la que reportaría el uso de un lenguaje de propósito general para ese mismo cometido, permitiendo acortar la distancia entre desarrolladores y usuarios [2].

Dado que los dos principios básicos de la MDE son potenciar el rol de los modelos y elevar el nivel de automatización [1], no sólo se necesitan nuevos lenguajes de modelado, sino también herramientas que permitan facilitar el procesamiento

y definición de modelos. Así, de la misma forma que los programadores cuentan con IDEs que les ayudan en las tareas de codificación, los desarrolladores necesitan entornos de modelado que les asistan cuando trabajan con un DSL.

El auge de la MDE ha impulsado la aparición de herramientas para la producción de este tipo de entornos. La mayoría adoptan una aproximación basada en el metamodelo, en la que primero se define la sintaxis abstracta del lenguaje que luego se utiliza como base para la generación de la sintaxis concreta, editores, validadores, motores de transformaciones, etc., hasta que se dispone de un entorno completo para trabajar con uno o varios DSLs [3,4].

Desafortunadamente, la adopción de una aproximación basada en el metamodelo contribuye a acrecentar uno de los problemas habituales de las herramientas MDE actuales: todos los principios de la Interacción Persona-Ordenador (*Human-Computer Interaction*, HCI) [5] han sido prácticamente ignorados hasta la fecha a la hora de construir nuevas soluciones. En particular, la definición de sintaxis concretas (visuales) ha consistido básicamente en la asignación arbitraria de símbolos gráficos a los conceptos que componían la sintaxis abstracta del nuevo lenguaje.

Este escenario se explica porque, al igual que ha pasado antes con otras disciplinas emergentes, en sus inicios el objetivo de proporcionar soluciones que funcionen para probar que la nueva disciplina merecía atención y reconocimiento ha prevalecido sobre la consideración de aspectos relacionados con la calidad de esas soluciones. En realidad es otro ejemplo de la clásica dicotomía entre el *time-to-market* y la calidad. Sin embargo, dado que la MDE ha alcanzado ya ciertos niveles de madurez [6], ha llegado el momento de empezar a considerar aspectos de calidad en el desarrollo de nuevas herramientas y propuestas dirigidas por modelos [7].

Con la intención de contribuir en esta línea, el presente trabajo introduce `CEViNEdit`, una herramienta EMF (*Eclipse Modeling Framework*) que descansa en los principios de Moody [8] para soportar el desarrollo de editores gráficos para DSL visuales, teniendo en cuenta la velocidad, facilidad y precisión con que las notaciones visuales pueden ser procesadas por la mente humana (lo que se conoce como *eficacia cognitiva* [9]). Para ello, `CEViNEdit` adopta también una aproximación basada en el metamodelo, pero ayuda a los desarrolladores en el proceso de asignar representaciones gráficas a los conceptos que lo componen de acuerdo a la teoría de la Física de las Notaciones (*Physics of Notations*, PoN) [8]. La idea subyacente es que el diseño de notaciones visuales y la selección de elementos gráficos debe basarse (en la medida de lo posible) en sólidas bases teóricas y cierta evidencia empírica de su eficacia cognitiva, en lugar de limitarse a seguir algunas buenas prácticas y convencionalismos [10].

El resto del artículo se estructura de la siguiente forma: en la sección 2 se presenta la motivación y el marco de investigación de este trabajo; la sección 3 presenta `CEViNEdit`, una herramienta que permite evaluar y tener en cuenta algunos aspectos relacionados con la eficacia cognitiva durante el desarrollo de editores gráficos que dan soporte a los DSLs; por último, en la sección 4 se presentan las conclusiones derivadas de nuestro trabajo.

2. Marco de investigación

Esta sección presenta la motivación del presente trabajo. Para ello, en lugar de la habitual revisión de trabajos relacionados, se analizan someramente las principales características de las propuestas que permiten la generación de editores gráficos. Dicho análisis revela que, a pesar de haber alcanzado ciertos niveles de madurez, no se suelen emplear mecanismos que permitan tener en cuenta la calidad de las notaciones visuales soportadas por los editores gráficos generados.

A continuación presentamos un breve análisis de las principales propuestas teóricas que se pueden utilizar para dar solución a este problema. Dicho análisis sirve para respaldar la elección de la Física de las Notaciones de Moody [8] como base teórica para desarrollar una herramienta que permita generar editores gráficos que tengan en cuenta la eficacia cognitiva de las notaciones visuales.

2.1. Herramientas para el desarrollo de editores gráficos

En los últimos años, bajo la certidumbre de que la MDE se está acercando a la rampa de convencimiento del habitual ciclo de sobreexpectación que experimenta toda propuesta tecnológica [11], hemos desarrollado diferentes trabajos orientados a la adopción de aproximaciones más rigurosas y sistemáticas para el desarrollo de herramientas MDE [12,13] y la aplicación de los principios de la MDE a otros campos [14].

Una de las principales lecciones aprendidas de estos trabajos es que probablemente la característica más atractiva de la MDE para investigadores de otras áreas es la posibilidad de expresar sus problemas en términos de modelos elaborados en términos de abstracciones que les resulten familiares. Esto implica obviamente el desarrollo de nuevos DSLs ajustados a sus necesidades y la construcción de sus correspondientes editores.

Por tanto, con el fin de conocer cuáles eran las posibilidades a la hora de construir estos editores, siguiendo las directrices de Biolchini et al. [15] para el desarrollo de revisiones sistemáticas en el contexto de la Ingeniería del Software, hemos llevado a cabo una revisión de la literatura de las herramientas que permiten generar editores gráficos a partir de un modelo de dominio. Así, en lugar de presentar una revisión clásica de los trabajos relacionados, optamos a continuación por presentar los principales resultados de dicha revisión sistemática.

Las preguntas de investigación que nos planteamos al inicio de la revisión fueron las siguientes:

- ***PI1***: ¿Existen herramientas que apliquen los principios de la MDE para generar editores gráficos?
- ***PI2***: ¿Cuáles son las principales características y funcionalidades de dichas herramientas?

De esta forma, y tomando como punto de inicio las herramientas de metamodelado previamente presentadas en [13], añadimos algunas otras herramientas y las analizamos según los criterios presentados a continuación. Cabe resaltar

que esta sería una tarea sin fin, dado que cada día surgen nuevas propuestas. Por ejemplo, Sirius, desarrollada por los responsables de Obeo Designer, es una herramienta que no se había presentado aún cuando abordamos esta revisión.

Ámbito. La herramienta es comercial o de código abierto. En este último caso se menciona el tipo de licencia.
Sintaxis abstracta. Notación o lenguaje usado para la especificación de la sintaxis abstracta.
Sintaxis concreta. Forma de asignar un símbolo gráfico a los conceptos que componen la sintaxis abstracta.
Separación conceptual entre sintaxis. La herramienta trata de forma separada la sintaxis abstracta y concreta.
Capacidades de edición. Valoración subjetiva de la modificabilidad de los editores generados.
Uso de modelos. Valoración del nivel de utilización de modelos para el desarrollo del editor gráfico.
Automatización. Nivel de automatización.
Usabilidad. Este criterio aúna diferentes aspectos relacionados con la usabilidad, como la facilidad de uso o la documentación disponible.
Integración. La herramienta se ejecuta sobre un framework existente o es autónoma.
Base científica. La herramienta sigue o aplica algún tipo de principio o método para guiar, derivar o definir notaciones visuales. Este criterio está en cierta forma subsumido en el de usabilidad, pero consideramos oportuno mostrarlo por separado con el fin de ilustrar claramente el problema abordado por la herramienta que se presenta en este trabajo.

Debido a las limitaciones de espacio, la Tabla 1 sólo muestra los valores correspondientes a algunas de las herramientas. Se ha optado por incluir aquellas que se entienden mejor alineadas con los criterios mencionados anteriormente. [1].

Como se puede observar en la Tabla 1, la mayor parte de las propuestas analizadas resuelven el problema de la separación conceptual entre la sintaxis abstracta y la notación visual, evitando así contaminar el metamodelo. Asimismo, algunas herramientas permiten automatizar muchos de los pasos intermedios del desarrollo de un editor, pero la mayoría necesitan de refinamientos manuales a través de mecanismos complejos y poco intuitivos. Se ha podido comprobar que la mayoría están construidas sobre Eclipse, lo que asegura ciertos niveles de interoperabilidad con otras herramientas MDE existentes. Finalmente, la conclusión más importante desde el punto de vista de este análisis, es que a pesar de que existen herramientas dirigidas por modelos para el desarrollo de editores gráficos relativamente maduras y estables, ninguna de ellas tiene en cuenta cuestiones de calidad en relación con las notaciones visuales de los editores generados. Por lo tanto, a continuación revisamos algunas de las propuestas teóricas existentes para el análisis de notaciones visuales y presentamos brevemente la que finalmente seleccionamos para construir una nueva herramienta que mejore el estado del arte.

[1] La tabla completa puede consultarse en `http://www.kybele.es/cevinedit`

Tabla 1. Vista general de herramientas para la generación de editores gráficos

Criterios	*DiaGen*	*Eugenia*	*GMF*	*MetaEdit+*	*Obeo*
Ámbito	OS(GPL)	OS(EPL)	OS(EPL)	Com	Com
Sintaxis abstracta	Ecore/UML	Ecore	Ecore	GOPPRR	Ecore
Sintaxis concreta	DiaMetaDesign	EOL	Draw2D	API interna	Odesign
Distinción sintaxis	No	Sí	Sí	Sí	Sí
Edición	✔✔	✔✔	✔✔✔	✔✔✔	✔✔✔
Modelos	✔✔	✔✔✔	✔✔✔	✔✔	✔✔✔
Automatización	✔✔	✔✔✔	✔✔	✔✔✔	✔✔✔
Usabilidad	✔✔✔	✔✔✔	✔✔	✔✔✔	✔✔✔
Integración	Eclipse	Eclipse	Eclipse	None	Eclipse
Base científica	No	No	No	No	No

Leyenda: *Pobre* (✔), *Bueno* (✔✔), *Excelente* (✔✔✔)

2.2. Propuestas para el análisis de notaciones visuales

Históricamente la Ingeniería del Software ha prestado mayor interés a la definición de métodos para evaluar la semántica de un lenguaje y ha infravalorado los temas relacionados con su sintaxis concreta. Así, podría afirmarse que la definición de notaciones visuales en Ingeniería del Software está tradicionalmente basada en cuestiones subjetivas, ignorando casi por completo aspectos como la comunicación visual, el estudio de los signos, el diseño gráfico o la cognición. Probablemente el ejemplo más claro de esta situación sea el lenguaje UML, que a pesar de ser un estándar mundialmente adoptado, no proporciona ninguna documentación sobre las decisiones que condujeron a la definición de su notación visual. Peor aún, la tendencia histórica a considerar UML casi como la única opción válida para soportar tareas de modelado, ha hecho que la forma de definir su sintaxis concreta haya sido erroneamente (veáse [16] por ejemplo) adoptada por gran parte de los lenguajes que han aparecido con el auge de la MDE, convirtiendo las `cajas-y-flechas` en estándar de-facto.

Entre las pocas propuestas aparecidas hasta la fecha que podrían utilizarse para evaluar una sintaxis concreta, las más relevantes son *Cognitive Dimensions* (CD's) [17], Semiotic Quality (SEQUAL) [18] y Physics of Notation (PoN) [8]. Cada una de estas propuestas define, o podría llegar a utilizarse para definir un marco de trabajo para soportar la definición y evaluación de notaciones visuales.

SEQUAL está basada en teoría semiótica y proporciona una lista de propiedades para evaluar la calidad de modelos y lenguajes de modelado, para lo cual define una completa ontología que recoge las dimensiones de la calidad de los lenguajes de modelado. En su estado actual, SEQUAL dista de estar orientada al trabajo con notaciones visuales y por ello la descartamos como base teórica para este trabajo.

Por otro lado, la propuesta de las *Cognitive Dimensions* (CDs), presentada inicialmente como un conjunto de características que permiten comparar la forma y estructura de los lenguajes de programación, ha sido probablemente la

propuesta más referenciada por los investigadores que trabajan en la usabilidad de los lenguajes visuales. Sin embargo, la literatura afirma que las CDs adolecen de ciertos problemas desde el punto de vista del presente trabajo, algunos de los cuales son incluso apuntados por los propios autores:

- Fue diseñado para utilizarse en cualquier dominio, desde hojas de cálculo hasta lenguajes de programación [19].
- Las definiciones poco precisas de las dimensiones y la ausencia de un procedimiento para aplicar la propuesta, complican su comprensión y uso [20].
- El número de dimensiones ha ido creciendo notablemente, hasta alcanzar un número inmanejable para usuarios que no hayan participado en su definición.

Teniendo en cuenta las limitaciones de estas propuestas, entendemos que el marco más apropiado para el presente trabajo es la Física de las Notaciones de Moody [8], una propuesta definida exclusivamente para diseñar, evaluar, comparar y mejorar notaciones visuales. No en vano, con el auge de la MDE, en los últimos años han sido varios los trabajos que la han utilizado para evaluar notaciones visuales de importantes lenguajes como UML [16], BPMN [21] o WebML [22]. La siguiente sección introduce brevemente esta propuesta.

2.3. La Física de las Notaciones

La PoN establece nueve principios para definir, comparar y evaluar notaciones visuales. Cada uno de estos principios se deriva de teorías y evidencias empíricas traídas de diferentes disciplinas de la comunicación visual, como la psicología perceptiva y cognitiva, el diseño gráfico o incluso la cartografía.

Cada principio contiene: estrategias de diseño para mejorar las notaciones visuales; una serie de métricas para comparar las diferentes notaciones; y ejemplos de notaciones que satisfacen o violan el principio. Su objetivo es que la creación de notaciones visuales deje de ser un proceso inconsciente para convertirse en un proceso consciente que mejore la calidad de la comunicación visual.

A continuación se enumeran los nueve principios de esta teoría sin entrar en los detalles de su definición:

1. Principio de la **Claridad Semiótica**: debería existir una correspondencia 1:1 entre los conceptos semánticos y los símbolos gráficos.
2. Principio de la **Discriminabilidad Perceptiva**: diferentes símbolos deben ser claramente distinguibles uno del otro.
3. Principio de la **Transparencia Semántica**: utilizar representaciones visuales que sugieran el significado del concepto que representan.
4. Principio de la **Gestión de la Complejidad**: incluir mecanismos explícitos para gestionar la complejidad.
5. Principio de **Integración Cognitiva**: incluir mecanismos explícitos para soportar la integración de información de diferentes diagramas.
6. Principio de **Expresividad Visual**: utilizar toda la gama y capacidades de las variables visuales.
7. Principio de **Codificación Dual**: usar texto para complementar los gráficos.

8. Principio de **Economía Gráfica**: el número de símbolos gráficos diferentes debería ser cognitivamente manejable.
9. Principio de **Ajuste Cognitivo**: utilizar diferentes dialectos visuales para diferentes tareas y públicos.

A pesar de las ventajas proporcionadas por la PoN, si nos centramos exclusivamente en la mejor forma de representar visualmente un conjunto de elementos, esta propuesta también tiene algunas limitaciones. Ninguno de los principios de la PoN permite valorar la efectividad de las reglas de composición de un lenguaje, lo que podría suponer la generación de diagramas ineficaces a nivel cognitivo.

Sin embargo, conviene resaltar que las reglas de composición provienen mayoritariamente de la definición de la sintaxis abstracta del lenguaje, la cual precede al desarrollo de la sintaxis concreta en la habitual aproximación dirigida por el metamodelo. En respuesta a este problema han aparecido trabajos incipientes para que el desarrollo de un DSL parta de la definición de su notación [23]. Estas aproximaciones dirigidas por la notación encajan mejor con el marco que definen las CDs, ya que facilita el replanteamiento de las reglas de composición del lenguaje. Por contra, la PoN encaja mejor con la tradicional aproximación dirigida por el metamodelo. Aunque utilizar la PoN no asegura que el DSL desarrollado sea un lenguaje cognitivamente eficaz, sí proporciona cierto nivel de confianza sin comprometer el balance entre esfuerzo y resultado.

Para confirmar esta suposición, en pasados trabajos utilizamos la PoN para evaluar la notación visual de WebML implementada por WebRatio [24,22], un IDE basado en Eclipse para soportar el desarrollo dirigido por modelos de aplicaciones Web [25]. Una de las lecciones aprendidas más importantes que extrajimos de este trabajo es que, aunque la evaluación de la eficacia cognitiva de un DSL visual es factible en términos de tiempo y esfuerzo, resolver los problemas detectados durante la evaluación es una tarea prácticamente imposible o demasiado costosa.

Por lo tanto, los aspectos de usabilidad del lenguaje deberían ser contemplados lo más pronto posible, de manera que las consideraciones al respecto se trasladasen a lo largo de todo el proceso de desarrollo hasta el resultado final.

Para asistir al desarrollador en este sentido, en este trabajo presentamos nuestra propuesta para soportar el desarrollo de editores para DSL visuales que tengan en cuenta la eficacia cognitiva, es decir, la velocidad, facilidad y precisión con que las notaciones visuales pueden ser procesadas por la mente humana [9].

2.4. Variables Visuales

Antes de presentar nuestra propuesta, debemos introducir otro concepto: las `Variables Visuales`, que se emplean en muchos de los principios de Moody. Estas variables constituyen los bloques básicos con los que codificar información de forma gráfica. El trabajo más importante en este área es el de Bertin [26], considerado para el diseño gráfico lo que la tabla periódica para la química.

Tal y como muestra la Fig. 1, las variables visuales pueden agruparse en dos subconjuntos: planares y retinianas. Cada una de estas variables tiene una

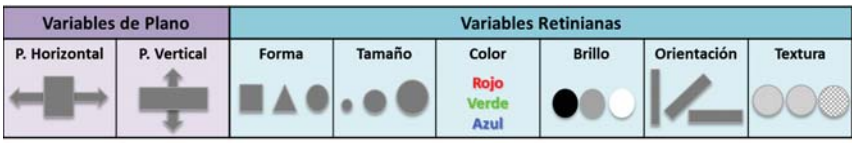


Fig. 1. Variables visuales usadas para construir notaciones visuales (adaptada de [26])

serie de propiedades que pueden ser utilizadas para codificar un cierto tipo de información. Es fundamental conocer estas propiedades con el fin de elegir correctamente el rango de valores apropiado para cada variable y de esta forma generar notaciones eficaces a nivel cognitivo.

3. CEViNEdit: automatizando y mejorando el desarrollo de notaciones visuales

En esta sección presentamos `CEViNEdit`, nuestra propuesta para generar editores gráficos para DSLs visuales. Esta herramienta proporciona un desarrollo focalizado en la automatización de todo el proceso de creacón de un editor gráfico y en la producción de notaciones visuales cognitivamente eficaces gracias al marco teórico propuesto por Bertin [26] y Moody [8].

Para conseguir tal fin, hemos utilizado un enfoque de desarrollo dirigido por modelos, con el objetivo de aumentar el nivel de abstracción y disminuir los tiempos de desarrollo. De esta forma, hemos desarrollado en primera instancia un DSL sobre el que se basa nuestra herramienta. En esencia, este DSL abstrae dos conceptos: los elementos gráficos que pueden usarse como bloques de construcción a la hora de diseñar editores basados en GMF (`NodeEClass`, `LinkEReferences`, etc.) y las variables visuales definidas por Bertin [26]. Los modelos generados conformes a este metamodelo permiten almacenar la información relacionada con la representación gráfica de los elementos del modelo de dominio.

Una vez diseñado nuestro DSL, utilizamos las librerías de EMF para generar un plugin genérico sobre el cual fuimos desarrollando toda nuestra herramienta, además de las librerías SWT y JFace para generar una interfaz dinámica que se ajustara a nuestras necesidades. Asimismo, para la parte relacionada con la personalización gráfica de los editores gráficos generados, `CEViNEdit` basa su funcionamiento en las anotaciones especificadas por EuGENia por un lado, y en la generación de ficheros Java y EOL por otro. Así pues, para definir la sintaxis abstracta de un modelo de dominio, se hace un uso mixto de las anotaciones de ambas técnicas mediante una serie de librerías creadas ad-hoc. Por otro lado, para la evaluación de la eficacia cognitiva de la notación visual creada por el usuario, hemos creado unas librerías propias con el fin de evaluar algunos de los principios de la PoN. Por último, para el apartado de la generación automática del editor, `CEViNEdit` utiliza la tecnología proporcionada por Epsilon, la cual a partir de un modelo de dominio con anotaciones de EuGENia junto a ficheros EOL, nos permite generar un editor GMF personalizado y de forma automática.

3.1. Proceso de desarrollo de un editor gráfico con CEViNEdit

En esta sección describiremos brevemente la interfaz de usuario de CEViNEdit y su proceso de desarrollo para generar editores basados en GMF. Para ello, utilizaremos como caso de estudio el ejemplo simple e intuitivo empleado por EuGENia [2].

La Fig. 2 muestra la interfaz de usuario proporcionada por CEViNEdit. Dicha interfaz consta de 4 paneles y una barra de herramientas. El primer panel (A) es un editor a forma de árbol que muestra el modelo de dominio del cual se quiere generar un editor gráfico. Este panel permite navegar por todos los elementos del modelo de dominio y seleccionar solo aquellos elementos que se deseen personalizar. El segundo panel (B) muestra el modelo de CEViNEdit, el cual se encarga de almacenar las relaciones necesarias entre los elementos de la sintaxis abstracta y la sintaxis concreta con el fin de generar el editor basado en GMF. El tercer panel (C) es una serie de controles dinámicos que varían en función del tipo de elemento seleccionado en el modelo de CEViNEdit. Este panel permite establecer los valores de las variables visuales de los elementos gráficos que compondrán la notación visual. El cuarto panel (D), muestra contenidos de ayuda dinámicos relacionados con la influencia de la variable visual seleccionada sobre la eficacia cognitiva. Por último, la barra de herramientas superior (E) permite evaluar la notación visual definida por el usuario según algunos de los principios de la PoN y cuenta además con el control que permite lanzar todo el proceso de generación del editor gráfico.

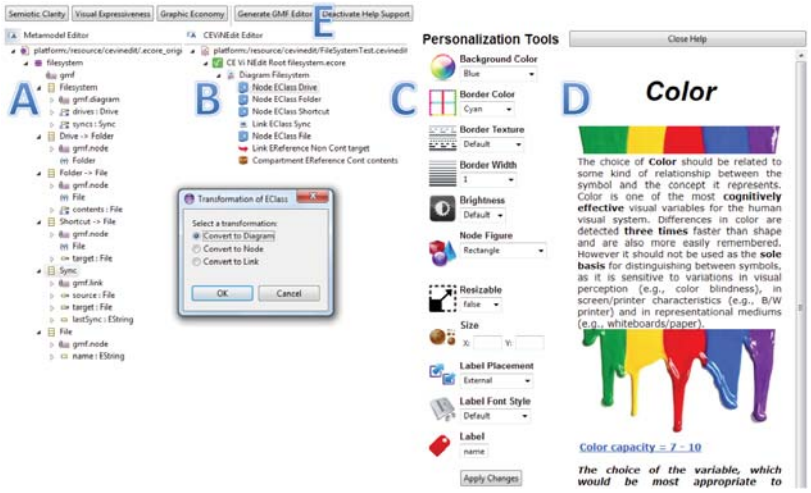


Fig. 2. Interfaz de usuario de CEViNEdit

En cuanto al proceso de desarrollo de un editor gráfico utilizando nuestra propuesta, el primer paso es crear un modelo CEViNEdit dentro de un proyec-

[2] EuGENia GMF Tutorial: http://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/

to general de Eclipse. La primera interfaz de usuario es un simple panel desde el que se debe especificar la ruta donde se encuentra el modelo de dominio, el cual puede estar en el workspace del mismo Eclipse o en cualquier ubicación del sistema de ficheros del ordenador. Posteriormente, el usuario debe elegir los elementos gráficos de GMF que representarán cada uno de los elementos del modelo de dominio. El asistente de la herramienta tiene en cuenta las restricciones de GMF, con lo cual muestra sólo los posibles valores en base al elemento del dominio seleccionado. Estas decisiones se almacenan en el modelo de `CEViNEdit` presente en el segundo panel. Teniendo en cuenta el caso de estudio mencionado anteriormente, la `EClass` denominada *filesystem* será representada por un `gmf-diagram`; las `EClass` *File*, *Drive*, *Folder* y *Shortcut* se representarán mediante un `gmfnode` y finalmente, la `EReference` *Shortcut.target* y la `EClass` *Sync* se mostrarán como `gmflinks`.

Según el tipo de elemento seleccionado en el modelo `CEViNEdit` (Node, Link o Compartment), el tercer panel muestra una lista de variables visuales con desplegables que contienen sus posibles valores. De esta forma, el usuario puede definir las principales características de la sintaxis concreta mediante anotaciones con un alto nivel de abstracción y de una forma simple e intuitiva. Para facilitar aún más esta tarea, el cuarto panel muestra una ayuda dinámica cada vez que el usuario selecciona una variable visual. Esta ayuda contiene información relacionada con el trabajo de Moody y Bertin [8,26] con el fin de que el usuario pueda realizar una elección apropiada para cada uno de los valores de las variables visuales disponibles. Una vez terminada una primera versión de la notación visual definida por el usuario, es posible ejecutar la valoración automática de dicha notación según algunos de los principios de la Física de las Notaciones.

Finalmente, se puede invocar la generación del editor basado en GMF. A partir de este momento, todo el proceso se realiza de forma automática. Dicho proceso conlleva una serie de transformaciones internas que traducen las decisiones tomadas por el usuario en anotaciones conformes a GMF. Estas transformaciones permiten mantener una distinción conceptual entre la sintaxis abstracta y la sintaxis concreta, impidiendo una contaminación sobre el modelo de dominio original. A continuación se invoca el proceso de desarrollo de EuGENia paralelamente a diferentes transformaciones de sincronización propias de `CEViNEdit` con el fin de obtener un editor completamente funcional. Para una mejor visualización de todo el proceso, hemos creado un screencast disponible en `http://kybele.es/cevinedit`.

3.2. Valoración automática de los principios de Moody

Tal y como hemos mencionado anteriormente, la versión actual de `CEViNEdit` permite llevar a cabo una valoración automática de las decisiones almacenadas en el modelo `CEViNEdit` de acuerdo a tres de los nueve principios establecidos en la PoN. En la Fig. 2.E se puede observar el panel con los botones que permiten esta evaluación automática. Para cada uno de ellos la herramienta abre una ventana emergente con el informe de la evaluación de cada uno de los tres principios.

El primer principio que se puede evaluar es la **Claridad Semiótica**, el cual establece que según la teoría de símbolos de Goodman [27], pueden existir cuatro tipos de anomalías respecto a la correspondencia entre los elementos de un lenguaje y sus respectivos símbolos gráficos:

- La *Redundancia* implica que varios símbolos gráficos se utilizan para representar el mismo elemento del lenguaje (sinógrafos).
- La *Sobrecarga* implica que el mismo símbolo gráfico se utiliza para representar diferentes elementos del lenguaje (homógrafos).
- El *Exceso*, que se da cuando un símbolo gráfico no representa ningún elemento del lenguaje.
- El *Déficit*, que indica que un elemento no es representado por ningún símbolo gráfico.

El informe generado por `CEViNEdit` mostrará los resultados relacionados con este tipo de anomalías.

Por otro lado, con `CEViNEdit` es posible evaluar también la **Expresividad Visual**, la cual se define como el número de variables visuales utilizadas eficientemente por una notación visual. En general, las notaciones usadas en la Ingeniería del Software suelen utilizar un rango limitado de dichas variables. Por ejemplo, de entre los posibles valores de la variable Forma, el valor rectángulo es el más utilizado con diferencia por la mayor parte de los lenguajes de modelado. No obstante, los rectángulos no son muy eficientes si tenemos por objetivo facilitar el proceso de reconocimiento visual. Sería más conveniente utilizar otros tipos de formas, como curvas, formas 3D o directamente iconos. Además, de acuerdo a investigaciones en la psicofísica, cada una de las ocho variables visuales tiene una capacidad, es decir, el número de los niveles que la mente humana puede distinguir. Así pues, la elección de la variable visual usada y su respectivo valor, no debe ser una decisión arbitraria y sí una decisión relacionada con el tipo de información que se quiere codificar. En este contexto, el informe generado por `CEViNEdit`, muestra el número de valores usados para cada variable visual, su capacidad y su saturación, es decir, la correlación entre el número de valores utilizados y la capacidad de cada variable, lo cual indica hasta qué punto la variable está siendo utilizada de forma eficiente.

Por último, `CEViNEdit` permite evaluar el principio de la **Economía Gráfica**, el cual establece que la estrategia de proporcionar un cierto número de símbolos en un lenguaje visual es efectiva hasta que el proceso de reconocimiento cognitivo se convierte en una tarea excesivamente compleja. La capacidad humana para discriminar entre las diferentes alternativas percibidas en su memoria de trabajo consiste aproximadamente de seis categorías [28]. Este número es, por tanto, un límite superior para la complejidad gráfica. Generalmente, los lenguajes de modelado tienden a incrementar su complejidad gráfica en un esfuerzo para mejorar su expresividad semántica, es decir, la eficacia de un modelo para reflejar la realidad intrínseca que ese modelo representa. De este modo, `CEViNEdit` informa al usuario acerca de la complejidad gráfica de la notación visual que se está diseñando.

3.3. Limitaciones y trabajos futuros

Con el fin de validar nuestra propuesta, sería conveniente llevar a cabo un estudio empírico que ya hemos planificado con estudiantes de la rama de informática. De este modo podremos evaluar no sólo la usabilidad de la herramienta, sino también cualquier aspecto que pueda ser obtenido a partir de usuarios finales con el fin de tenerlos en cuenta para mejoras en las próximas versiones. Además se está llevando a cabo un trabajo para poder evaluar automáticamente otros principios de la teoría de la Física de las Notaciones. Sin embargo, cabe resaltar que no todos los principios son aptos a evaluarse según el diseño de nuestra propuesta. Por ejemplo, la Integración Cognitiva se aplica sólo cuando múltiples diagramas se utilizan para representar un solo sistema. Este tipo de problemática va más allá del alcance de nuestra aplicación, dado que este principio evalúa la eficacia cognitiva de sistemas en los que se emplean diferentes tipos de diagramas heterogéneos. Del mismo modo, estamos trabajando para poder proporcionar un mayor número de anotaciones de GMF que se puedan utilizar en la herramienta. Para ello estamos ampliando nuestras librerías para generar código EOL automáticamente. Finalmente, consideramos que dado que ya obtenemos automáticamente algunas métricas relacionadas con la eficacia cognitiva de una notación visual, nos encontramos en una buena posición para desarrollar mecanismos que permitan mejorar dichas métricas y plasmarlas sobre la notación visual de forma automática. Del mismo modo, consideramos oportuno el desarrollo de nuevos mecanismos que nos permitan evaluar la eficacia cognitiva sobre notaciones visuales desarrolladas previamente a partir de otras herramientas como GMF o EuGENia. Dicha evaluación se debería desarrollar sobre los modelos internos de GMF como el `gmfgraph`, `gmfmap` y `gmftool`.

4. Conclusiones

En este trabajo hemos presentado `CEViNEdit`, una herramienta que soporta el desarrollo dirigido por modelos de editores gráficos orientada a producir notaciones visuales cognitivamente eficaces de acuerdo a los principios de la PoN [8].

Hasta donde sabemos, `CEViNEdit` es la primera herramienta que considera aspectos de calidad/usabilidad en el desarrollo dirigido por modelos de editores gráficos. No obstante, entendemos que es este un área en la que aún queda mucho trabajo por hacer, dado que hasta la fecha la mayor parte de estas herramientas han sido construidas por desarrolladores técnicamente muy capaces pero con poca experiencia en temas relacionadas con la cognitividad o la interacción persona-computador [5].

Conviene resaltar que el uso de `CEViNEdit` no basta para garantizar que la notación visual del editor producido sea cognitivamente eficaz, pero sí asegura que su definición descansa al menos en parte en sólidos principios teóricos y proporciona por tanto ciertos niveles de confianza. En cierto modo, la idea es similar a la que guía las pruebas del software: la cobertura total, o bien no es factible o el esfuerzo que requiere no compensa los beneficios que reporta.

Más allá de la funcionalidad que proporciona la herramienta, con este trabajo pretendemos además potenciar el interés de la comunidad MDE (DSDM en España) en utilizar buenas prácticas y sólidas bases teóricas a la hora de definir un lenguaje visual. En el fondo, esta idea es parte del movimiento que algunos miembros de la comunidad han comenzado ya a hacer en favor de incluir consideraciones de calidad en el desarrollo de nuevas propuestas [7].

En este sentido, este trabajo ha servido para mostrar que no sólo es factible automatizar ciertos aspectos relacionados con la inclusión de aspectos de calidad en el desarrollo de herramientas MDE, sino que además es posible hacerlo manteniendo el compromiso entre esfuerzo y tiempo dedicado. Esta observación es particularmente relevante si tenemos en cuenta que hasta ahora los miembros de la comunidad MDE se han centrado en demostrar la aplicación práctica de sus propuestas, prestando poca atención a los aspectos relacionados con la calidad. Si queremos que empiecen a tener en consideración aspectos de calidad, debemos proporcionarles herramientas que les permitan hacerlo sin que esto suponga demasiado esfuerzo adicional en el proceso de desarrollo. En otras palabras, la automatización es clave para compensar el incremento en términos de complejidad accidental que conlleva la inclusión de aspectos de calidad.

Agradecimientos. Esta investigación ha sido llevada a cabo en el marco del proyecto MASAI (TIN-2011-22617), financiado por el Ministerio de Ciencia e Innovación.

Referencias

1. Schmidt, D.C.: Guest editor's introduction: Model-driven engineering. IEEE Computer **39**(2) (2006) 25–31
2. Ghosh, D.: Dsl for the uninitiated. Queue **9**(6) (June 2011) 10:10–10:21
3. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven Software Engineering in Practice. Synthesis digital library of engineering and computer science. Morgan & Claypool Publishers (2012)
4. Gronback, R.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Eclipse Series. Pearson Education (2009)
5. Whittle, J., Hutchinson, J., Rouncefield, M.: Industrial Adoption of Model-Driven Engineering : Are the Tools Really the Problem ? In: Model-Driven Engineering Languages and Systems, Lecture Notes in Computer Science Volume 8107, MoDELs 2013. (2013)
6. Volter, M.: From programming to modeling-and back again. Software, IEEE **28**(6) (2011) 20–25
7. Selic, B.: What will it take? a view on adoption of model-based methods in practice. Software and System Modeling **11**(4) (2012) 513–526
8. Moody, D.L.: The "physics.ºf notations: a scientific approach to designing visual notations in software engineering. In: ICSE (2). (2010) 485–486
9. Zhang, J., Norman, D.A.: Representations in distributed cognitive tasks. Cognitive Science **18**(1) (1994) 87–122
10. Wheildon, C., Ogilvy, D., Heard, G.: Type and Layout: Are You Communicating Or Just Making Pretty Shapes? Worsley Press (2005)

11. Linden, A., Fenn, J.: Understanding gartner's hype cycles. Strategic Analysis Report N° R-20-1971. Gartner, Inc (2003)
12. Bollati, V.A., Vara, J.M., Jiménez, A., Marcos, E.: Applying MDE to the (semi-)automatic development of model transformations. Information and Software Technology **55**(4) (2013) 699 – 718
13. Vara, J.M., Marcos, E.: A framework for model-driven development of information systems: technical decisions and lessons learned. Journal of Systems and Software **85**(10) (2012) 2368–2384
14. Vara, J.M., Andrikopoulos, V., Papazoglou, M.P., Marcos, E.: Towards model-driven engineering support for service evolution. Journal of Universal Computer Science **18**(17) (2012) 2364–2382
15. Biolchini, J., Gomes, P., Cruz, A., Horta, G.: Systematic review in software engineering. Technical Report ES 679/05 (2005)
16. Moody, D.L., van Hillegersberg, J.: Evaluating the visual syntax of uml: An analysis of the cognitive effectiveness of the umlfamily of diagrams. In: SLE. (2008) 16–34
17. Green, T.R.G.: Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay (Eds.) People and Computers V.Cambridge, UK: Cambridge University Press (1989) 443–460
18. Krogstie, J., Sindre, G., Jørgensen, H.D.: Process models representing knowledge for action: a revised quality framework. EJIS **15**(1) (2006) 91–102
19. Green, T.R., Blandford, A.E., Church, L., Roast, C.R., Clarke, S.: Cognitive dimensions: Achievements, new directions, and open questions. Journal of Visual Languages & Computing **17**(4) (2006) 328–365
20. Dagit, J., Lawrance, J., Neumann, C., Burnett, M., Metoyer, R., Adams, S.: Using cognitive dimensions: advice from the trenches. Journal of Visual Languages & Computing **17**(4) (2006) 302–327
21. Genon, N., Heymans, P., Amyot, D.: Analysing the cognitive effectiveness of bpmn 2.0 visual notation. In Malloy, B.A., Staab, S., van den Brand, M., eds.: SLE. Volume 6563 of Lecture Notes in Computer Science., Springer (2010) 377–396
22. Granada, D., Vara, J.M., Brambilla, M., Bollati, V., Marcos, E.: Analysing the cognitive effectiveness of the webml visual notation. Software and Systems Modelling (Revised and Resubmitted)
23. Wouters, L.: Towards the Notation-Driven Development of DSMLs. In Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P., eds.: Model-Driven Engineering Languages and Systems, Lecture Notes in Computer Science Volume 8107, MoDELs 2013, Springer (2013) 522–537
24. Granada, D., Vara, J.M., Brambilla, M., Bollati, V., Marcos, E.: Analysing the cognitive effectiveness of the webml visual notation. Technical report, School of Computer Science, Rey Juan Carlos University (2013)
25. Acerbis, R., Bongio, A., Brambilla, M., Butti, S.: Webratio 5: An eclipse-based case tool for engineering web applications. In: ICWE. (2007) 501–505
26. Bertin, J.: Semiology of Graphics - Diagrams, Networks, Maps. ESRI (2010)
27. Goodman, N.: Languages of Art: An Approach to a Theory of Symbols. Bobbs-Merrill Co. (1968)
28. Miller, G.: The magical number seven, plus or minus two: Some limits on our capacity for processing information. The Psychological Rev **63** (1956) 81–97

SEPL: Social Environment Programming Language

Luis-María García-Rodríguez, Álvaro Gutiérrez-Pérez, Rober Morales-Chaparro, Fernando Sánchez-Figueroa

Quercus Software Engineering Group, Universidad de Extremadura
luismaex|agutierrez|robermorales|fernando@unex.es

Resumen. En los últimos años el uso de redes y aplicaciones sociales ha colonizado muchos procesos en organizaciones. Esto ha derivado en falta de automatización de los mismos, y en la necesidad de conectarse manualmente a diferentes redes para realizar estas tareas repetitivas. Proponemos SEPL, un DSL que se conecta con varias redes y aplicaciones sociales, para permitir automatizar estos procesos.

Palabras clave: Desarrollo de software dirigido por modelos, Lenguajes específicos del dominio, Redes sociales

1. Introducción

Vivimos en una sociedad altamente interconectada en la que a diario trabajamos con redes sociales (Twitter, Facebook, etc) como aplicaciones sociales (Correo, Google Drive&Docs, Dropbox, Doodle, etc). Si bien el ámbito de uso personal es uno de los grandes beneficiados de esta oleada de herramientas, sin duda alguna, es en el ámbito profesional donde se está produciendo uno de los saltos más importantes.

Sin embargo la creciente proliferación de aplicaciones sociales ha propiciado una gran **dispersión** que ha conllevado que el usuario tenga que publicar en muchos casos el mismo contenido en repetidas ocasiones en todas las aplicaciones sociales en las que participa (**redundancia**) o que deba hacer una y otra vez las mismas tareas(**falta de reutilización**). Cada vez que el usuario quiere trabajar en una red social diferente a la que está debe cambiar de contexto, distrayéndose, y por tanto se ralentiza y le genera una **mala experiencia de usuario**. En este artículo mostramos, a través de un ejemplo director (Sección 2) la ocurrencia de estos problemas y cómo algunas herramientas existentes tratan de solventarlo (Sección 3). En la Sección 4 mostramos SEPL mientras que la Secciones 5, 6 y 7 se muestran algunos detalles de implementación, evaluación, conclusiones y trabajos futuros.

2. Motivación

El caso de estudio elegido responde a una situación real que se da a la hora de toma de decisiones en un ámbito político. Consideremos el procedimiento a la hora de publicar una nota de prensa. Este proceso consta de las siguientes etapas:

1. El responsable de comunicación redacta una nota de prensa y la envía a través de un email a la junta directiva para que la revise.
2. Espera a que al menos 3 de los miembros contesten dando un OK.
3. A continuación se publica la nota dentro de un grupo de Linkedin en el que hay una serie de afiliados y simpatizantes, los cuales pertenecen a una comisión que se encarga de detectar posibles problemas que haya. Si estos miembros están de acuerdo usarán 'I like' para indicar su aprobación.
4. Una vez tenga 30 'I likes' se entiende que la nota es apta para su publicación.
5. Finalmente se envía a los medios de comunicación y se publicita a través de Twitter, o cualquier otra red social elegida.

Quien realice esta función debe estar pendiente del correo, revisar la publicación en Linkedin hasta que llegue al número de 'I likes', enviar a los medios la nota de prensa y publicitar la nota en todas sus redes sociales. En este proceso se detectan los problemas señalados en la Sección 1.

3. Trabajos relacionados

El objetivo de esta sección es mostrar una panorámica de las herramientas que existen para tratar de solucionar los problemas identificados anteriormente, compararlas atendiendo a varias características y demostrar la pertinencia de SEPL. El conjunto de herramientas no es ni mucho menos exhaustivo, pero sí representativo. Otros autores [1,2] han intentado antes solucionar problemas similares. Además, proyectos de investigación como *BPM4people*[1] abordaron la inclusión de estos mecanismos en la definición de procesos de negocio. Sin embargo, en nuestro conocimiento, la propuesta de un DSL con su implementación que trabaje con varias redes sociales no ha sido realizada con anterioridad. Se han buscado características que suponen la contraposición a los problemas de la Sección 1. Entendiendo que una herramienta que cumpla con todas ellas solucionará la problemática.

- El uso de programación gráfica o textual. La mayoría de las herramientas usan programación gráfica, renunciado a la flexibilidad otorgada por una **notación textual (C1)**.
- **Sencillez (C2)** a la hora de poder implementar un caso de uso. Prácticamente todas son sencillas de utilizar.
- Provisión de **filtros avanzados (C3)** en las operaciones, es decir la posibilidad de acotar el problema incluyendo condiciones o requisitos como por ejemplo contar el número de 'I likes'.

[0] BPM4people: www.bpm4people.org

- Provisión de **estructuras de programación avanzadas (C4)** que nos permitan establecer condiciones y acciones complejas. Desde el punto de vista de la lógica computacional, no todos los problemas a los que los usuarios se van a enfrentar son de tipo 'if condition then action'.
- Los usuarios valoran tener una **experiencia positiva (C5)** con una herramienta, sobre todo si la **curva de aprendizaje es rápida (C6)** y la herramienta es **intuitiva (C7)**. Son aspectos subjetivos, pero no por ello menos importante y de los que puede depender que una herramienta triunfe.
- Finalmente se ha incluido una columna para destacar si la herramienta **mantienen el contexto (C8)** durante la ejecución, es decir si permite focalizar sobre un contenido en concreto e individual una acción.

Tabla 1. Comparativa de funcionalidades en las herramientas actuales.

	C1	C2	C3	C4	C5	C6	C7	C8
IFTTT		✓			✓	✓	✓	
Zapier		✓			✓	✓	✓	
We Wired Web		✓			✓	✓	✓	
Elastic.io		✓			✓	✓	✓	
Cloud Work		✓			✓	✓	✓	
WappWolf[1]		✓			✓	✓	✓	✓
itDuzzit			✓	✓				
Yahoopipes			✓	✓				
Kissflow[2]			✓		✓			✓
SEPL	✓	✓	✓	✓	✓	✓	✓	✓

Tras la elaboración de esta tabla comparativa, se concluyó que cualquier intento de dar una solución avanzada más allá de la idea *if-condition-then-action* estaba sin desarrollar, y las pocas herramientas que prometían solucionarlo, como itDuzzit, no estaban implementadas y daban soluciones tan poco intuitivas que terminaban abandonadas por los usuarios. Por tanto era pertinente desarrollar SEPL, un lenguaje específico del dominio [3] que permitiera implementar casos de uso avanzados, en los que se interactúe con redes sociales, que sean reutilizables y que incluya estructuras de programación típicas, para que cualquier usuario que haya programado previamente en algún lenguaje pueda aprender con rapidez a usar SEPL.

4. SEPL

A continuación describimos las principales características de SEPL mediante el uso de pequeños ejemplos.

[1] sólo es válido para interactuar con Dropbox
[2] sólo es válido para las herramientas en la nube de Google

Fácil de usar: Se utiliza un lenguaje que se aproxima mucho al lenguaje natural, con lo que simplemente leyendo el código se entiende fácilmente lo que está haciendo. Concretamente a continuación se expone la propuesta que soluciona el caso de uso en la publicación de una nota de prensa, en Fig. 1.

```
1  send Email(juntadirectiva@googlegroups.com) body "Nota de prensa" = $1
2      wait response('Ok',3) then
3          send Linkedin (group "Afiliados y simpatizantes") body $1
4              wait Ilikes(30) then
5                  send Email(medioA@unex.es & medioB@unex.es) body $1
6                  send Twitter body "Nota de prensa" = $1
```

Figura 1. Caso de estudio

Redes sociales configurables: El usuario puede decidir en tiempo de programación con qué red social va a trabajar, Fig. 1@3,6, pero si no se tiene claro o simplemente se quiere reutilizar la receta para varios casos, tendrá la posibilidad de decidirlo en tiempo de instanciación (Fig. 2), inmediatamente anterior a tiempo de ejecución.

```
send SocialNetwork body "Nota de prensa" = $1
```

Figura 2. Caso red social genérica

APPs sociales: A parte de redes sociales, se pueden usar muchos otros servicios, mail, la nube de Google, Dropbox, encuesta Doodle, etc. Fig. 3.

```
send Email(hola@sepl.es) body "Mi email \n Estoy seguro que os gustará SEPL"
send GoogleSpreadsheet("/miTabla") body $msg1
```

Figura 3. Caso apps sociales

Variables globales: No se repiten tampoco los contenidos de los mensajes en la receta, para ello se usarán variables globales del sistema que se definen con $ (Fig. 1@1). De esta forma evita perder el contexto y ataja el problema de la poca reutilización y repetición de información.

Enviar mensajes públicos, privados o a grupos: Una de las principales funciones de las actuales redes sociales consisten justo en publicar mensajes en el timeline, en los grupos o páginas, o mensajes privados. Todo ello también está recogido dentro de SEPL, Fig. 1@3,6. Si quisiéramos enviar un privado podríamos hacerlo como en la Fig. 4

```
send Twitter(@spel) body "Este es un mensaje privado de prueba"
send Linkedin("David Martín Rodríguez") body "Mensaje privado"
```

Figura 4. Enviar mensajes

Programar eventos: Una de las principales ventajas de SEPL es poder automatizar y programar eventos. De esta forma no hay que estar pendiente de un evento para lanzar otro, se puede programar bajo qué condiciones ejecutamos una nueva acción.

Usando la expresión correcta se consigue que el hilo de ejecución espere uno o varios eventos antes de continuar, ello es gracias a la expresión wait, como puede verse en la Fig. 1@2,4.

Detectar y contar adhesiones: 'I like' es una de las grandes invenciones de Facebook y de las redes sociales que se le asemejan. Hasta ahora ninguna herramienta de las evaluadas permitía usar las adhesiones, contarlas, manejar los grupos de usuarios, etc. Como se puede ver en la Fig. 1@4, se puede contar el número de likes de una publicación y usarlo como condición.

Detectar respuestas: Automatizar también el proceso de respuesta es importante, no vale cualquier publicación o cualquier correo, debe ser la respuesta para no perder el contexto y trabajar sobre un mismo hilo. El objetivo no es hacer un análisis del texto de la respuesta y por tanto su complejidad queda fuera de alcance, pero sí se puede plantear un pequeño análisis de ciertas palabras, es lo que sucede en la Fig. 1@2.

Responder a mensajes: Al igual que se detectan las respuestas, se puede responder automáticamente, si así se define, a los mensajes que lleguen a las redes sociales, ya sean públicos o privados, o se trate de un email como se ve en la Fig. 5.

```
wait Email('usuario@spel.es') then reply "Gracias por la rápida respuesta"
```

Figura 5. Responder a mensajes

5. Implementación

El flujo que sigue la ejecución (Fig. 6) es muy sencillo. Cada vez que un programador escribe código SEPL este se transforma gracias a Xtend a una estructura simple como es YAML. Este documento es recogido e interpretado por nuestro parser, escrito en Ruby, y es en tiempo de ejecución cuando el intérprete interactuará con las APIs de las aplicaciones sociales, y en caso de que haga falta (por necesidad de credenciales o instanciación) también lo hará con los usuarios. Cuestiones como el polling son resueltas en el parser.

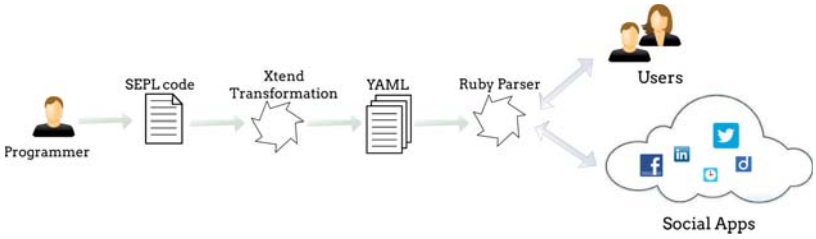


Figura 6. Flujo de ejecución

6. Evaluación

Llegado este punto se evaluó el trabajo realizado a través de un test a varios estudiantes de la Universidad de Extremadura. En primer lugar, resolvieron una serie de ejemplos y el caso de estudio de forma tradicional, paso a paso con cada una de las redes y aplicaciones sociales. Posteriormente con las herramientas actuales analizadas en la Sección 3 y finalmente con SEPL. El resultado obtenido fue el esperado, en el primero de los casos el tiempo fue alto, en el segundo no consiguien más que automatizar pasos concretos y poco eficiente, y con SEPL consiguieron con una curva de aprendizaje bastante pequeña, proponer recetas para solucionar el caso de estudio incluso mejoras a la propuesta que se hacía.

Si bien el rendimiento puede parecer un problema a priori, no lo es ya que no se tiene que analizar continuamente el estado del servicio y todo lo que sucede, sino la ocurrencia de un evento sobre el contenido que publicamos.

7. Conclusiones y trabajo futuro

Este ambicioso proyecto conlleva mucho e importante de trabajo futuro asociado. Como incluir más aplicaciones sociales para conseguir abarcar el mayor número de ellas. Desarrollar un sitio web para empezar a probar a gran escala el dominio del lenguaje y detectar así problemas con los que los usuarios puedan encontrarse. Implementar una interfaz gráfica para ofrecer una alternativa a los usuarios que no necesiten o no quieran usar la programación textual.

Referencias

1. Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. The jabberwocky programming environment for structured social computing. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 53–64. ACM, 2011.
2. Juan Jara, Florian Daniel, Fabio Casati, and Maurizio Marchese. From a simple flow to social applications. In *Current Trends in Web Engineering*, pages 39–50. Springer, 2013.
3. Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

Proceso de verificación de reglas de transformación basado en métricas*

Fernando Macías, Roberto Rodríguez-Echeverría, Víctor M. Pavón, José M. Conejero, and Fernando Sánchez-Figueroa

Universidad de Extremadura,
Quercus Software Engineering Group
{fernandomacias, rre, victorpavon, chemacm, fernando}@unex.es
http://quercusseg.unex.es

Resumen. La Ingeniería Dirigida por Modelos (IDM) se basa fundamentalmente en la definición de metamodelos, la edición de modelos y la especificación de transformaciones entre éstos (MMT, *Model-to-Model Transformations*). En muchos casos el desarrollo, evolución y adaptación de estas transformaciones se sigue realizando sin apoyarse en métodos o herramientas que permitan reducir el esfuerzo y los costes asociados a estas actividades. En este trabajo se presenta un proceso que permite medir el grado en que las reglas que implementan dichas transformaciones se adecúan a su especificación. Para ello, se plantea el proceso de verificación de MMT como un proceso de extracción de información aplicándose métricas ampliamente utilizadas en este tipo de escenarios. Este trabajo de verificación de MMT se ha desarrollado y probado en la adaptación de reglas de transformación dentro del proyecto MIGRARIA.

Palabras clave: MDD, transformación, métricas, MDSD.

1 Introducción

El presente trabajo se enmarca dentro del proyecto MIGRARIA [1], que define un proceso de modernización de aplicaciones Web heredadas (AWHs) hacia clientes móviles y *Rich Internet Applications* (RIA). En la fase de ingeniería inversa de este proceso, se define un análisis estático de los artefactos software de la AWH mediante MMT con el fin de abstraer su información y estructurarla conforme a un metamodelo propio independiente de la tecnología: MIGRARIA MVC. Con el objetivo de reutilizar este análisis estático en otros casos de estudio, ha sido necesario adaptar o extender las reglas de transformación definidas para que sean capaces de soportar nuevos patrones de código, puesto que las guías de estilo de código utilizadas en el desarrollo de la AWH diferían de las contempladas previamente. En este contexto, surgió la necesidad de disponer

* Trabajo financiado por el Ministerio de Ciencia e Innovación, Proyecto TIN2011-27340; la Consejería de Empleo, Empresa e Innovación del Gobierno de Extremadura, Ayuda GR10129; y el Fondo Europeo para el Desarrollo Regional (FEDER)

de un proceso que ayudase a realizar la adaptación de la reglas señalando, por un lado, qué reglas faltaban o no estaban funcionando como se esperaba y, por otro lado, permitiendo comprobar de manera rápida que las reglas adaptadas funcionaban conforme a su especificación.

En este trabajo utilizamos el concepto de verificación definido en [2] como "la correspondencia entre la especificación y la implementación en un producto software", para definir el problema presentado como un problema de verificación, en este caso de reglas de transformación modelo a modelo. La aproximación seguida en este trabajo parte de la premisa de que el problema planteado es asimilable a un problema de extracción de información y, por lo tanto, se pueden utilizar métricas bien conocidas dentro de ese ámbito [3]. Las tres métricas que se utilizarán en este planteamiento son: *precisión*, *exhaustividad* y *valor-F*, obtenidas para cada tipo de elemento que aparezca en el modelo de salida de las MMT. Para calcularlas, se partirá de datos estadísticos obtenidos mediante consultas sobre los modelos de entrada y salida. A partir de estos datos, se deducirán una serie de valores que, finalmente, se podrán utilizar para calcular las métricas. El análisis de estas métricas permitirá realizar la verificación de las MMT.

En resumen, en este trabajo planteamos un proceso de verificación de reglas de transformación mediante métricas, el cual sirve como apoyo a la hora de realizar la adaptación de reglas de transformación. Este proceso puede dividirse en dos fases: (1) la obtención de las métricas, que permiten la verificación de reglas de transformación mediante su interpretación, y (2) el uso de esta información para guiar el proceso de adaptación de las reglas. Este artículo se centra en la primera fase, ofreciendo una vista general del proceso de obtención y haciendo hincapié en la interpretación de los valores de las métricas que se obtienen como resultado de la aplicación de dicho proceso. El escenario de aplicación de partida, como ya se ha expuesto, es el proceso de ingeniería inversa definido en MIGRARIA, sin descartar su aplicación en otros escenarios. En este contexto, nos interesa verificar el correcto funcionamiento de las MMT, así como comprobar que el proceso de ingeniería inversa ha extraído correctamente toda la información que se esperaba. Las métricas planteadas nos permiten satisfacer el segundo objetivo y, en esta propuesta, queremos valorar su adecuación para la verificación de MMTs, como alternativa a las propuestas realizadas hasta ahora.

El resto de este artículo se divide de la siguiente forma. La sección 2 comenta algunos trabajos relacionados con éste. La sección 3 presenta la propuesta. Y la sección 4 apunta las conclusiones de este artículo y el trabajo futuro a realizar.

2 Trabajos relacionados

En esta sección se comentan otras técnicas para la verificación de las reglas de transformación con respecto a su especificación. La terminología utilizada para clasificarlas se ha obtenido de [4] y [5].

En primer lugar, encontramos técnicas de verificación formal basadas en el uso de restricciones OCL. En esta línea se encuentra [6], donde se combinan los modelos de entrada y salida, las transformaciones ejecutadas y las restricciones

OCL sobre ellos en un único modelo, sobre el que pueden definirse postcondiciones OCL que sirvan para encontrar contraejemplos en caso de error en la especificación. También definen planteamientos similares, haciendo uso de OCL, los trabajos de [7], que define restricciones en forma de *Tracts*, como una alternativa de carácter más generalista que la presentada en [8], la cual se centra en el *testing* de transformaciones mediante oráculos. De forma similar, en la propuesta presentada en [9] también se opta por un cambio de representación: se traducen reglas ATL y restricciones OCL sobre los metamodelos implicados a *First Order Logic*, que luego son verificados mediante solucionadores SMT.

En segundo lugar, también se han encontrado trabajos que definen técnicas informales para verificar la correspondencia estructural de los modelos de entrada y salida, como [10], centrado en generar avisos de error en el código de las reglas de transformación a partir de la especificación.

Según los criterios de los surveys mencionados al principio de esta sección, nuestra propuesta se clasifica como un método directo e informal para verificar la correspondencia estructural. Cabe destacar que, en este área, ninguno de los trabajos revisados aborda el problema de la verificación de reglas de transformación utilizando métricas, por lo que en la siguiente sección se presenta un método que parte de este planteamiento.

3 Propuesta

A continuación se ilustran las dos partes del proceso presentado: (1) la obtención de las métricas y (2) su interpretación.

3.1 Proceso de obtención de las métricas

Este proceso se ha implementado de forma manual en ATL sobre los casos de estudio utilizados en MIGRARIA. En este trabajo, por cuestiones de espacio, se ilustra mediante el conocido ejemplo *Families2Persons* [11]. La implementación en ATL de este ejemplo se encuentra accesible públicamente[1].

Los cinco pasos que conforman el proceso son:

1. **Selección de elementos del modelo de destino**: Los valores de las métricas se calculan para cada uno de los tipos de elemento en el modelo destino. En el ejemplo *Families2Persons*, estos tipos son *Male* y *Female*.
2. **Establecimiento de relaciones entre elementos de ambos modelos**: El segundo paso consiste en identificar todos los elementos en el modelo de entrada a partir de los cuales se genera cada uno de los tipos de elemento destino seleccionados en el paso anterior, información que puede obtenerse de la especificación de las reglas. Las dos relaciones detectadas en este ejemplo son entre *Male* y *Member* y entre *Female* y *Member*, ambas con cardinalidad 1:1 (un elemento de entrada genera un elemento de salida en ambos casos).

[1] `https://bitbucket.org/migraria/families2persons/src`

3. **Definición de criterios de exclusión**: Estos criterios deben servir para identificar los elementos del modelo de entrada que no deben ser transformados. En el ejemplo *Families2Persons*, se puede definir, por ejemplo, la exclusión de "todos los *Members* del modelo de entrada que no pertenezcan a ninguna familia". En escenarios con modelos y transformaciones de mayor volumen y complejidad, estos criterios deben extraerse de fuentes documentales concretas, como por ejemplo guías de estilo de código en MIGRARIA.
4. **Definición de criterios de detección**: Estos criterios deben permitir comprobar si un elemento se ha transformado, es decir, encontrar los elementos que deberían haberse generado en el modelo destino a partir de un elemento de origen dado. Por ejemplo, para identificar un *Female* generado a partir de un *Member*, una comparación de nombre y apellido resulta suficiente. Esta información es posible extraerla también de la traza de la ejecución de las transformaciones, si está disponible.
5. **Cálculo de las métricas**: Finalmente, puede procederse a calcular los valores de precisión, exhaustividad y valor-F para cada elemento del modelo destino. Para ello, deben utilizarse los conjuntos de elementos (1) *positivos*, que cumplen los criterios de detección y no los de exclusión; (2) *falsos positivos*, que cumplen los criterios de exclusión y de detección; y (3) *falsos negativos*, que no cumplen los criterios de exclusión ni de detección. Las fórmulas para realizar este cálculo son:

$$Precisión = \frac{p}{p+fp} Exhaustividad = \frac{p}{p+fn} F_1 = 2 \cdot \frac{Pr \cdot Ex}{Pr + Ex}$$

La Tabla 1 muestra los valores obtenidos en el ejemplo utilizado. En este caso, se detectan falsos positivos en la generación de elementos *Male* que afectan a la precisión, puesto que en el modelo de entrada existe un elemento *Member* de género masculino sin un elemento *Family* asociado. En el resto de casos, los valores de las métricas son perfectos.

Tabla 1. Resumen del proceso de obtención de las métricas

Elemento origen	Elemento destino	Criterios exclusión	Excluidos	P	FP	FN	Precisión	Exhaustividad	Valor-F
Male	*Member (father/son)*	Sin familia	1	5	1	0	83.33 %	100.00 %	0.91
		Total	1						
Female	*Member (mother/daughter)*	Sin familia	0	4	0	0	100.00 %	100.00 %	1.00
		Total	0						

3.2 Interpretación de los resultados

Una vez calculadas las métricas se realiza el análisis de los resultados, el cual permite detectar errores o carencias en las reglas de transformación utilizadas. Puesto que éste es un paso fundamental de la propuesta, a continuación se incluye una relación de algunos de los resultados que más información pueden aportar en este sentido, junto con su interpretación:

- **Errores que afectan a la precisión**. Son causados por un falso positivo. Apuntan a un error en la selección de los elementos origen: se ha transformado un elemento que no debería haberlo hecho, como el caso comentado anteriormente entre *Male* y *Member*. Los valores de las métricas sugieren adecuar el filtro en la regla de transformación que genera elementos *Male* para evitarlo.
- **Errores que afectan a la exhaustividad**. Son causados por un falso negativo. Pueden indicar que la regla de transformación encargada de generar un tipo de elemento está incompleta o excluye indebidamente algunos elementos del origen, puesto que no se ha ejecutado una transformación que sí debería haberlo hecho.
- **Errores que afectan a ambas métricas simultáneamente**. Las dos métricas son independientes, por lo que la existencia de errores simultáneos en ambas indica simplemente que sus causas también ocurren a la vez: errores de filtrado y transformaciones incompletas. También puede indicar falta de expresividad del metamodelo destino.
- **Imposibilidad de calcular precisión**. En algunos casos, puede ser imposible calcular el valor de la precisión (y por tanto del valor-F), en los cuales además el valor de la exhaustividad es 0 %. Se debe a que no existen elementos clasificados como positivos ni como falsos positivos, por no haberse transformado ningún elemento (exhaustividad 0 %). La explicación más factible es que no existen reglas de transformación definidas para el caso dado.
- **Imposibilidad de calcular exhaustividad**. No existen elementos clasificados como positivos ni como falsos negativos: ningún elemento de entrada del tipo dado debe transformarse, de acuerdo con la especificación. No se necesitan, por tanto, reglas de transformación. Una combinación de este caso con la posibilidad de calcular la precisión, señalaría que las reglas de transformación están definidas, pero sólo generan falsos positivos. Por lo tanto, se sugiere eliminar dichas reglas.

4 Conclusiones y trabajo futuro

Este trabajo presenta un proceso de verificación de reglas de transformación mediante métricas propias de la extracción de información. Este proceso ha sido utilizado con éxito en procesos de adaptación de reglas de transformación, que especifican análisis estáticos de código, dentro de MIGRARIA. En este contexto de adaptación, se ha propuesto una interpretación de los resultados de las métricas que se estima puede ser de utilidad para la verificación de reglas de transformación.

Una de las conclusiones de su aplicación en MIGRARIA es que nuestra propuesta parece ser más adecuada para la verificación de MMTs cuando se trabaja con modelos de gran tamaño y se requiere asegurar que el modelo destino contiene toda la información esperada. Otras propuestas relacionadas, como por ejemplo el uso de oráculos, pueden conseguir los mismos objetivos. Sin embargo,

creemos que nuestra propuesta debería ser más eficiente en términos de esfuerzo y beneficio. En este sentido, dentro de las líneas de trabajo futuro, el primer paso que planteamos es realizar experimentos, a partir de los casos de estudio de MIGRARIA, que nos permitan comparar nuestra propuesta con las ya existentes y ver si se satisface esta hipótesis. Otra de las líneas de trabajo futuro consiste en estudiar la necesidad y adecuación de definir un DSL para simplificar la especificación del cálculo de las métricas (definir los elementos, sus relaciones y los criterios de exclusión y detección) e independizarla del lenguaje final. También se pretende analizar diferentes lenguajes ya existentes que puedan jugar este rol o servir como lenguaje final.

Referencias

1. Roberto Rodríguez-Echeverría, José M. Conejero, Pedro J. Clemente, and Juan Carlos Preciado. Migración dirigida por modelos de sistemas web heredados a rich internet applications. In *XVI Jornadas de Ingeniería del Software y Base de Datos*, 2011.
2. Barry W Boehm. Verifying and validating software requirements and design specifications. In *IEEE software*. Citeseer, 1984.
3. George Forman. An extensive empirical study of feature selection metrics for text classification. *The Journal of machine learning research*, 3:1289–1305, 2003.
4. Moussa Amrani, Levi Lucio, Gehan Selim, Benoit Combemale, Jürgen Dingel, Hans Vangheluwe, Yves Le Traon, and James R Cordy. A tridimensional approach for studying the formal verification of model transformations. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on*, pages 921–928. IEEE, 2012.
5. Lukman Ab Rahim and Jon Whittle. A survey of approaches for verifying model transformations. *Software & Systems Modeling*, pages 1–26, 2013.
6. Gehan MK Selim, Fabian Büttner, James R Cordy, Juergen Dingel, and Shige Wang. Automated verification of model transformations in the automotive industry. In *Model-Driven Engineering Languages and Systems*, pages 690–706. Springer, 2013.
7. Martin Gogolla and Antonio Vallecillo. Tractable model transformation testing. In *Modelling Foundations and Applications*, pages 221–235. Springer, 2011.
8. J-M Mottu, Benoit Baudry, and Yves Le Traon. Model transformation testing: oracle issue. In *Software Testing Verification and Validation Workshop, 2008. ICSTW'08. IEEE International Conference on*, pages 105–112. IEEE, 2008.
9. Fabian Büttner, Marina Egea, and Jordi Cabot. *On verifying ATL transformations using 'off-the-shelf' SMT solvers*. Springer, 2012.
10. Nikolaos Drivalos Matragkas, Dimitrios S Kolovos, Richard F Paige, and Athanasios Zolotas. A traceability-driven approach to model transformation testing. In *AMT@ MoDELS*, 2013.
11. ATL Team. *ATL/Tutorials - Create a simple ATL transformation*. Online: `https://wiki.eclipse.org/ATL/Tutorials_-_Create_a_simple_ATL_transformation`. Accessed: 10-04-2014.

Primitive Operators for the Concurrent Execution of Model Transformations Based on LinTra

Loli Burgueño[1], Eugene Syriani[2], Manuel Wimmer[3], Jeff Gray[2], and Antonio Vallecillo[1]

[1] Universidad de Málaga, GISUM/Atenea Research Group, Málaga, Spain
{loli,av}@lcc.uma.es
[2] University of Alabama, Department of Computer Science, Tuscaloosa AL, U.S.A.
{esyriani,gray}@cs.ua.edu
[3] Vienna University of Technology, Business Informatics Group, Vienna, Austria
wimmer@big.tuwien.ac.at

Abstract. Performance and scalability of model transformations are becoming prominent topics in Model-Driven Engineering. In previous work, we introduced LinTra, a platform for executing out-place model transformations in parallel. LinTra is based on the Linda coordination language for archiving concurrency and distribution and is intended to be used as a middleware where high-level model transformation languages (such as ATL and QVT) are compiled. To define modularly the compilation, this paper presents a minimal, yet sufficient, collection of primitive operators that can be composed to (re-)construct any out-place, unidirectional model transformation language (MTL). These primitives enable any MTL to be executed in parallel in a transparent way, without altering the original transformation.

Keywords: Model Transformation, Primitives, LinTra

1 Introduction

Model-Driven Engineering is a relatively new paradigm that has grown in popularity in the last decade. Although there are a wide variety of approaches and languages with different characteristics and oriented to different types of model transformations (MT), most of the model transformation engines are based on sequential and local execution strategies. Thus, they have limited capabilities to transform very large models and even less capability to do it in a reasonable amount of time.

In previous works [1,2], we investigated concurrency and distribution for out-place transformations to increase their performance and scalability. Our approach, LinTra, is based on Linda [5], a mature coordination language for parallel processes. Linda supports reading and writing data in parallel into distributed tuple spaces. A tuple space follows the Blackboard architecture [3], which makes the data distributed among different machines transparent to the user.

To execute transformations on the LinTra architecture, LinTra specifies how to represent models and metamodels, how the trace links are encoded for efficient retrieval,

which agents are involved in the execution of the MT and their role, and how the transformation is distributed over the set of machines composing the cluster where the MT is executed. The implementation of several case studies using the Java implementation of LinTra (jLinTra) is available on our website [4], together with the performance comparison with several well-known model transformation languages (MTLs) such as ATL [6], QVT-O [8] and RubyTL [4].

The goal of this paper is to introduce a collection of minimal, yet sufficient, primitive operators that can be composed to (re-)construct any out-place and unidirectional MTL. These primitive operators encapsulate the LinTra implementation code in order to hide the underlying architecture that makes the parallel and distributed execution possible. This collection of primitives serves as an abstraction of the implementation details of the general-purpose language in which LinTra is implemented.

Section 3 introduces the collection of primitives. Section 4 illustrates examples of primitive combinations in order to write MTs. Finally, Section 5 presents our conclusions, the work that motivates our approach and an outlook on future work.

2 Background on LinTra

LinTra uses the Blackboard paradigm [3] to store the input and output models as well as the required data to keep track of the MT execution that coordinates the agents that are involved in the process.

One of the keys of our approach is the model and metamodel representation. In this representation, we assume that every entity in the model is independent from another. Each entity is assigned an identifier. Relationships between entities are represented by storing in the source entity the identifier of its target entity.

Traceability is frequently needed when executing an out-place model transformation because the creation of an element might require information about some other elements previously transformed, or even information about elements that will be transformed in the future. This means that there might be dependencies that can affect the execution performance, *e.g.*, when one element needs access to an element that has not been created yet. In LinTra, traceability is implemented implicitly using a bidirectional function that receives as parameter the entity identifier (or all the entity identifiers in the case that the match comprises more than one entity) of the input model and returns the identifier of the output entity(ies), regardless whether the output entities have already been created or not.

Together with the Blackboard, LinTra uses the Master-Slave design pattern [3] to run MTs. The master's job is to launch slaves and coordinate their work. Slaves are in charge of applying the transformation in parallel to submodels of the input model (partition) as if each partition is a complete and independent model. Since LinTra only deals with out-place transformations, the complete input model is always available. Thus, if the slaves have data dependencies with elements that are not in the submodels they were assigned, they only have to query the Blackboard to get them.

[4] http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTBenchmark

3 Collection of Primitives

3.1 Primitives

Primitive for the Concurrent Platform. LinTra requires the user to specify how the input model is partitioned. The `PartitionCreator` primitive receives the input model, an OCL expression, *OE*, and the maximum number of model entities, *S*, that each partition will contain. The `PartitionCreator` queries the input model using *OE* and partitions the resulting submodel into partitions of size *S*. The combination of `PartitionCreators` with different OCL expressions may lead to overlapping partitions; thus, the LinTra engine checks internally that the intersection of all the partitions is the empty set and the union is the whole model.

Primitives for the Model Transformation Language. The minimum set of primitive constructs needed to define out-place model transformations are: `Composer`, `Tracer`, `EntityCreator`, `CondChecker` and `Finder`.

`Composer` is a primitive that allows the grouping of a combination of primitives and assign the combination a name. Its syntax is *Composer* <*composerName*> { <*combination of primitives*> } and it is mainly used by the `Tracer`.

The `Tracer` provides access to the trace model needed by out-place MT engines for linking the entities in the output model. Given an input entity or set of entities that match the pre-condition of a rule, the traces give access to the entities that were created in the post-condition, and vice versa. In this case, to identify which primitive belongs to which rule, we propose to encapsulate them in a `Composer` so that the `Tracer` receives as parameter the name of the Composer and the set of entities from the pre or post-condition and give the reference to the other entities. Its signature is *Tracer(composer : Composer, e : Entity) : Collection(Entity)* and *Tracer(composer : Composer, e : Collection(Entity)) : Collection(Entity)*

`EntityCreator` creates an entity given its type and its features (attributes and bindings) and writes it in the Blackboard. The primitive receives as parameter the entity type and a dictionary which stores the values of every feature. Its syntax is *EntityCreator(type : Factory, features : Dictionary<feature, value>)*.

`CondChecker` allows the querying of the Blackboard with an OCL expression that evaluates to a boolean value. It receives as input the OCL expression, queries the Blackboard and returns the result. Its signature is *CondChecker(expr : OCLExpression) : Boolean*.

`Finder` allows the retrieval of elements from the Blackboard that satisfy a constraint. It receives as parameter an OCL expression and returns the set of entities (submodel) that fulfils the OCL expression. Its signature is *Finder(expr : OCLExpression) : Collection(Entity)*.

3.2 Integrating the primitives with the LinTra engine

When executing a transformation with LinTra there are several steps. Some of the steps are done automatically by the engine and others require that the user gives certain guidelines on how to proceed by means of the primitives. Two different phases can be distinguished: the setup and the MT itself.

The semantics of some MTs might require that a certain set of rules are applied to the whole input model before applying or after having applied some others. This is the case, for example, of top rules in QVT-R [8], and entrypoint and endpoint rules in ATL [6]. In order to be able to express this behaviour, in the setup phase, the rule schedule must be extracted from the transformation given by the user and a collection of collection of rules (collection of rule layers) must be created. All the rules belonging to the same layer can be executed in parallel, but all rules in one layer must have terminated before rules in a subsequent layer can begin.

Furthermore, during the setup, the transformation written in a high-level MTL is compiled to the MTL primitives, and the input model is parsed to the tuple space representation and stored into the Blackboard. Then, the `PartitionCreator` provided by the user is executed and the model partitions are created. Finally, the tasks to be executed by the slaves are created and stored in order in the Blackboard. A task is a pair consisting of a rule layer and a model partition. The tasks are produced by computing all the possible combinations between the partitions and the rule layers.

Once the setup phase is finished, the LinTra MT starts using the Master-Slave design pattern. The master creates slaves that execute the tasks that share the same rule layer and waits for all the tasks to be finished before starting to execute the ones that involve the following layer. Every slave executes the assigned task sequentially and all the slaves work in parallel. The master behaviour after launching the slaves is given by the pseudo-code presented in Listing 1.1.

Listing 1.1. Master.

```
1 rule_layer := 1
2 while (∃ task ∈ Blackboard.Tasks){
3     createSlaves()
4     while (∃ slave : slave.available and
5            ∃ task ∈ Blackboard.Tasks : task.ruleLayer = rule_layer) {
6         assign(slave, task) }
7     join() -- wait for all the slaves to finish
8     rule_layer := rule_layer+1 }
```

When a slave receives a task, it transforms the submodel given by its partition with the rules given by its rule layer. These rules are a collection of MT primitives. The code executed by the slaves is shown in Listing 1.2. An overview of how the system works can be seen in Figure 1.

Listing 1.2. Slave.

```
9 for each e ∈ task.partition { task.ruleLayer.transforms(e) }
```

4 Examples

Consider the metamodels for the Class-to-Relational case study described on our website [5] and a very simple transformation that from every Attribute creates a Column with the same name. From every non-abstract class creates a Table that is associated with the Columns that were created from its Attributes, where the name is the same as the Class name. Finally, and after having applied the previous rules, from every possible

[5] http://atenea.lcc.uma.es/index.php/Main_Page/Resources/MTBenchmark/Class2Relational

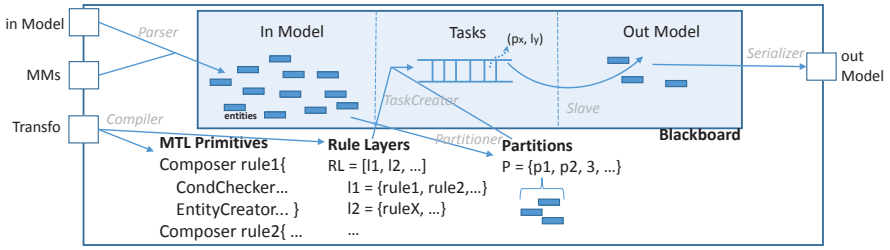


Fig. 1. Overview on the MT execution.

combination between Methods and Parameters, a Column must be created which name is the concatenation of the names of the elements from which it is created.

Let us assume that the user does not specify how the entities are assigned to the different partitions and the partition size is 100. The partition creator is invoked as *PartitionCreator(inModel, Entity.allInstances, 100)*. Let us suppose that it returns three partitions, $P = \{p1, p2, p3\}$. From the MT, the rule schedule is extracted and the rule layers are created. Given the MT definition, two different rule layers are created: $RL = [l1, l2]$ where *l1* contains the first two rules (class2Table and Att2Column) and *l2* the rule that is executed after the other two ruled have finished (MethParam2Column). Given the partitions and the layers, the tasks to be executed are $T = [T1, T2]$, where $T1 = \{(p1, l1), (p2, l1), (p3, l1)\}$ and $T2 = \{(p1, l2), (p2, l2), (p3, l2)\}$. We make the distinction between *T1* and *T2* to clarify that all tasks in *T1* are relative to *l1* and all tasks in *l2* to *T2*; thus, until all tasks from *T1* have been executed, tasks from *T2* cannot start. The compilation process from the high-level MT to the primitives produces the code shown in Listing 1.3

Listing 1.3. MTL primitives.

```
1  Composer ruleClass2Table { -- RuleLayer 1
2    if (CondChecker(e.oclIsTypeOf(Class) and not e.isAbstract))
3      EntityCreator(Table, {[name, e.name], [col, Tracer(ruleAtt2Column, e.att)]}) }
4  Composer ruleAtt2Column {
5    if (CondChecker(e.oclIsTypeOf(Attribute))
6      EntityCreator(Column, {[name, e.name]}) }
7  Composer ruleMethParam2Column { -- RuleLayer 2
8      if (CondChecker(e.oclIsTypeOf(Parameter))
9          meths := Finder(Method.allInstances)
10         for each (m : Method in meths)
11             EntityCreator(Column, {[name, e.name + '_' + m.name]}) }
```

The code for the first rule layer goes from line 1 to line 7. Lines 2, 5 and 9 define the `Composers`, one for each rule. Lines 3, 6 and 10 show the `CondCheckers` which impose the pre-conditions that the entities, e, have to fulfil for each rule. In line 7 the Columns are created using the `EntityCreator` in the same way that in line 4 the Tables are created, with the only difference is that in order to create the Tables, the `Tracer` is needed to create the links to its Columns. When the `CondChecker` in line 10 is fulfilled, the `Finder` in line 11 retrieves all the Methods in the Blackboard, and for each one, a Column is created in line 13.

5 Conclusion and Other Work

In this paper, we have presented a collection of primitives which will be combined for running concurrent and distributed out-place model transformations using LinTra. As this is our initial attempt to come up with the collection of primitives, in the next, we plan to study deeply existing MTLs and extend our set of primitive to encompass, some functionality that we are missing if any.

In terms of related work, several lines of work consider the transformation of large models and their performance and scalability problems [7]. With LinTra [1,2], and its current implementation written in Java, we provide a framework to execute parallel and distributed model transformations that requires all MTs to be executed in Java. With the goal of designing a Domain-Specific Language (DSL), we inspire our work on T-Core [9]. Specifically, on its collection of primitive operators that allows to write in-place MTs in an intermediate level of abstraction which is between the high-level MTLs and the low-level code used by the engines.

After we discover the complete set of primitive operators, there are some other lines of work we would like to explore. First, we will implement the primitives and encapsulate the LinTra code into them. To achieve that, we will explore how to formulate, in the most efficient way, the OCL constraints using the methods available in LinTra to query the Blackboard. Second, we plan to create compilers from the most common MTLs the primitives, so that distributed models can be transformed in parallel reusing MTs written in those languages by means of executing them in the LinTra engine. Third, we want to investigate some annotations for the high-level MTL, so that the user can provide the engine details such as how the parallelization must be done, how the input model should be partitioned, etc. to improve the performance of the transformation. Finally, we plan to investigate the possibility of creating a new and more specific high-level MTL.

Acknowledgments This work is funded by Research Project TIN2011-23795.

References

1. Burgueño, L.: Concurrent Model Transformations based on Linda. In: Doctoral Symposium @ MODELS. (2013)
2. Burgueño, L., Troya, J., Wimmer, M., Vallecillo, A.: On the Concurrent Execution of Model Transformations with Linda. In: BigMDE Workshop @ STAF. (2013)
3. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. Wiley, Chichester, UK (1996)
4. Cuadrado, J.S., Molina, J.G., Tortosa, M.M.: RubyTL: A Practical, Extensible Transformation Language. In: ECMFA. (2006)
5. Gelernter, D., Carriero, N.: Coordination languages and their significance. Commun. ACM **35**(2) (1992) 96–107
6. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming **72**(1-2) (2008) 31–39
7. Kolovos, D.S., Rose, L.M., Matragkas, N., Paige, R.F., Guerra, E., Cuadrado, J.S., De Lara, J., Ráth, I., Varró, D., Tisi, M., Cabot, J.: A research roadmap towards achieving scalability in model driven engineering. In: BigMDE Workshop @ STAF. (2013)
8. OMG: MOF QVT Final Adopted Specification. Object Management Group. (2005)
9. Syriani, E., Vangheluwe, H., LaShomb, B.: T-core: A framework for custom-built model transformation engines. Software & Systems Modeling (2013) 1–29

Propuesta de modelado de requerimientos en paradigmas de Ingeniería Web Ágil guiada por modelos*

J. Sedeño[1, 2], C.J. Torrecilla-Salinas[2], M.J.Escalona[2] and M. Mejías[2]

[1]Agencia Andaluza de Instituciones Culturales
jorge.sedeno@juntadeandalucia.es
[2]Grupo IWT2, Universidad de Sevilla
{jorge.sedeno,carlos.torrecilla}@iwt2.org
{mjescalona,risoto}@us.es

Resumen. El paradigma guiado por modelos ha sido utilizado en los últimos años para promover mejores resultados en el desarrollo de Aplicaciones Web, en el campo que se ha denominado Ingeniería Web Guiada por Modelos (MDWE, Model-Driven Web Engineering). Por otro lado se ha demostrado que las metodologías ágiles se adaptan de manera muy adecuada a los entornos web, al estar centradas en técnicas empíricas e iterativas sobre las necesidades de los usuarios y tener la flexibilidad adecuada a la hora de adaptarse a los cambios de los mismos y del entorno. En cualquier caso, el punto de partida de ambos son los requerimientos. Sin embargo, ambos puntos de partida tienen diferencias, ya que los requerimientos en técnicas ágiles difieren a los utilizados en los paradigmas MDWE debido a que estos últimos necesitan estar completamente definidos antes de ser transformados. El objeto del presente trabajo es proponer la estructura de un requerimiento ágil fundamentado en las Historia de Usuario, que pueda ser abordado usando las transformaciones de la ingeniería guida por modelo, intentando conjugar lo mejor de ambos paradigmas.

Palabras clave: Metodologías Ágiles, Ingeniería Web, Scrum, Model-Driven Web Engineering, Historia de Usuario.

1 Introducción

Desde hace algunos años las metodologías ágiles se han ido consolidado y extendiendo en organizaciones que desarrollan software en entornos Web [1], debido a que dan solución a las principales particularidades del paradigma de la Ingeniería Web, gracias a la flexibilidad que proporcionan. Estas características [2,3] podrían resumirse en:

- Entrega de valor "as soon as possible" y reducción del "time-to-market".
- Requisitos críticos en las interfaces persona-ordenador.
- Mutabilidad de los requisitos en cortos intervalos de tiempo y alta velocidad de adaptación a los cambios de requerimientos.
- Estructura de navegación compleja y aspectos específicos de seguridad.

* *Las opiniones expresadas en este artículo son las de sus autores, no siendo necesariamente compartidas por sus empleadores*

Por otra parte, la Ingeniería Web Guiada por Modelos (MDWE), se ha demostrado en los últimos años como un paradigma efectivo para la Ingeniería Web, que ha dado como resultado algunas propuestas válidas como UWE (UMLWeb Engineering) [4], WebML (Web Modeling Languages) [5] o NDT (Navigational Development Techniques) [6].

El presente artículo pretende modelar, basado en las recomendaciones de las metodologías ágiles más habituales; Scrum [8], eXtreme Programming (XP) [9], Crystal [10], Lean Software Development [11] o Feature Driven Development (FDD) [12], un requerimiento ágil.

Estas metodologías ágiles, al utilizar generalmente el concepto de requerimiento (necesidad), no lo hacen como los paradigmas tradicionales. La diferencia fundamental estriba en que las metodologías ágiles, que están basadas en procesos empíricos e iterativos, tienen como principal valor la agilidad, en una toma no detallada de requisitos y su posterior estimación y planificación [14, 15, 16]. Por el contrario, los paradigmas basados en metodologías tradicionales (en cascada, iterativas o incrementales) pivotan sobre una fase muy detallada de toma requisitos, que es necesaria para ejecutar las sucesivas transformaciones de sus modelos y pasar a las siguientes fases del desarrollo.

Actualmente hay investigaciones encaminadas a dotar de agilidad al paradigma MDWE, como los basados en mockups [18,19] o apoyados en la idea de llegar desde los requisitos a la ingeniería guiada por modelos en entornos ágiles, pero enfocándose en la transformación [20] y no en la definición del requerimiento ágil. En este artículo abordamos esta unión desde el punto de vista del modelado de requerimientos ágiles, para que puedan ser transformados posteriormente por métodos basados en el paradigma MDWE, del que no se han encontrado referencias en trabajos de investigación. Esto permitirá formalizar la toma de requerimientos ágiles en aras a poder realizar procesos de transformación que hagan todavía más eficiente la Ingeniería Web.

Este artículo está divido en tres secciones, la definición del requerimiento ágil en la sección 2, el ciclo de transformación ágil en la sección 3 y las conclusiones y trabajos futuros en la sección 4.

2 Requerimientos ágiles

Una "Historia de Usuario", término popularizado por Mike Cohn [7], es un conjunto acotado de funcionalidad que puede ser entregado a un proveedor, usuario o sistema de información, de manera que genera un valor que antes no existía, una vez que el Sprint ha concluido. Puede ser definido como una necesidad que genera valor por sí misma.

El ciclo de vida de los proyectos ágiles, en especial Scrum [8], comienza con la definición de una pila de elementos delimitados en un contexto, que darán forma al proyecto Web. Como tipo de elemento elegido se propone la Historia de Usuario, por su consolidación en el paradigma Ágil. Los elementos fundamentales del requerimiento ágil (Figura 1), que expondremos a continuación, están tomados de un marco ágil sobre las que se han desarrollado ya algunos proyectos [24,25]:

- La Historia de Usuario, conjunto de funcionalidad formada por un identificador univoco, una descripción que responderá a dicha funcionalidad, una fecha de alta, una valoración de negocio (unidades ficticias de negocio), una valoración de esfuerzo (unidades de esfuerzo ficticias, llamadas puntos de historia) y un elemento de ordenación y priorización, que en nuestra propuesta, será el retorno de la inversión (ROI), calculado como la división entre el valor de negocio y el esfuerzo, formarán el "core" del requerimiento ágil. La gran mayoría de técnicas ágiles incluyen estos elementos a excepción del ROI, que será el elemento clave en la elección de las historias de usuario a transformar, ya que nos proporcionará el marco espacio-temporal para su elección y que se ha demostrado muy eficaz [24, 25]. Las Historias de Usuario han de cumplir además los aspectos INVEST [9]: Independiente, Negociable, Valorable, Estimable, Acotado (Small), y Testeable.
- La Definición de Hecho, que en las metodologías ágiles, se entiende como un aserto que describe cuando una Historia de Usuario está terminada. Por tanto, una serie de definiciones de hecho constituirán los elementos que se transformarán en las pruebas, pudiendo utilizarse técnicas como Test Driven Development (TDD, [9]).
- Las restricciones, que ayudarán a delimitar la historia de usuario en un contexto concreto y que serán equivalentes a los requerimientos no funcionales.
- Las Personas, entidad análoga a los Actores en otros paradigmas, que tienen una relación con la Historia de Usuario. De estas relaciones entre la entidad Persona y la Historia de Usuario se deducirá el esquema de navegación Web. Se ha denominado Persona y no Actor al haber elegido el nombre de la técnica ágil Persona [19] dedicada al modelado de estas entidades y para no crear confusión con Actores cuya relación está basada en la causa-efecto, como por ejemplo, se da en un caso de uso.

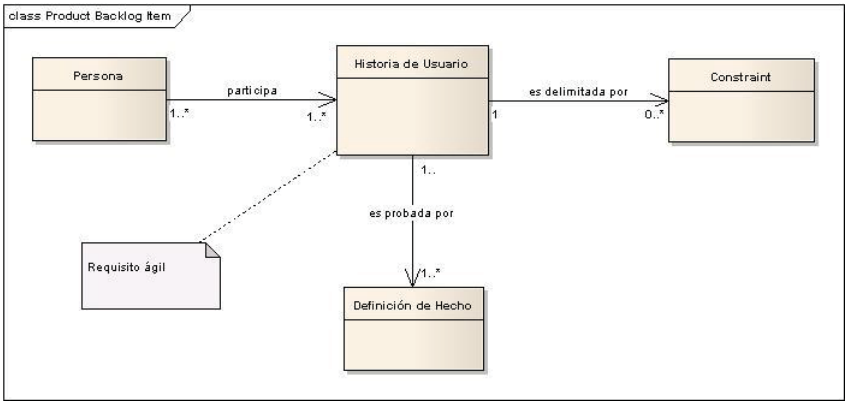


Fig. 1. Meta-modelo de la estructura de un Requerimiento Ágil

Existen técnicas ágiles para describir, trabajar y dar valor [21, 22, 23] a estos elementos y que se han desglosado en otros trabajos [24, 25], no siendo objeto del presente artículo.

Este meta-modelo conformaría un requerimiento ágil, formado por estas cuatro clases. Estos requerimientos ágiles serán los elementos de la pila de producto. Realmente representa un modelo de requerimiento que incorpora su propia priorización, de manera que podríamos hablar de entrega de valor priorizada o maximizada y por tanto ágil.

3 Ciclo de Transformación

Tomando como modelo Scrum [8], en el ciclo de vida de las metodologías ágiles, al diseño de la pila del producto, le siguen los correspondientes Sprints, de manera que sólo aquellas Historias de Usuario que han sido seleccionadas para el Sprint, es decir, aquellas cuyo ROI es mayor y por tanto su conclusión propiciará una mayor entrega de valor al cliente, serán las elegidas para ser transformadas, dando lugar a los elementos denominados *tareas, pruebas y navegación*. Una Historia de Usuario se descompondrá en una serie finita de *tareas*, las Definiciones de Hecho darán lugar a una serie de *pruebas* y las diferentes relaciones de las Personas con las Historias de Usuarios darán lugar a la navegación.

Partiendo del esquema que recomienda la OMG en su modelo sobre MDWE [13], estos los requerimientos ágiles, formarán el Modelo de Negocio (CIM, Computer Independent Model), en el cuál, se condensa la lógica de negocio. Estos requerimientos podrán ser transformados posteriormente, en el caso de técnicas ágiles, en cada sprint, en un modelo independiente de la plataforma (PIM, Platform Independent Model), como se observa en la Figura 2.

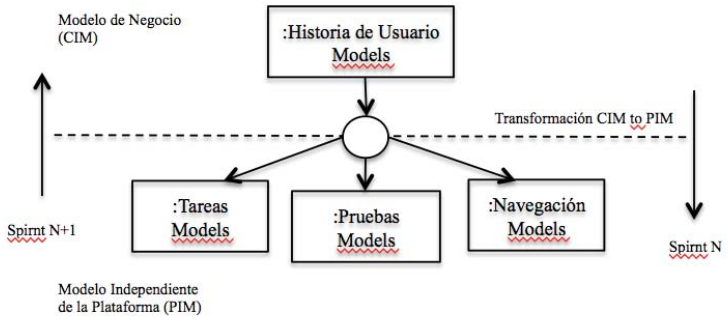


Fig. 2. Adaptación del modelo MWDE a metodologías ágiles

A diferencia del resto de paradigmas, que han de especificar y detallar todos los requisitos que luego son transformados, aquí solo se transformarán cada vez (en cada Sprint) un conjunto finito de requerimientos ágiles, pudiéndose refinar la pila del producto en cada sprint. En definitiva, sólo avanzan aquellos requerimientos necesarios para el momento concreto. Esto propiciará que la transformación sea incremental. Quedan fuera del artículo tanto los métodos de transformación como el paso al tercer

nivel, donde se dará una transformación a una tecnología específica (PSN, Platform Specific Model).

4 Conclusión y trabajos futuros

Esta modelo constituye una propuesta de formalización del concepto de requerimiento en proyectos basados en metodologías ágiles, tomando como base las Historias de Usuario, particularizando el concepto de requisito de las metodologías tradicionales, añadiendo la contextualización y el valor priorizado a entregar como parte integrante del mismo y como pieza fundamental del paradigma ágil. Es un requerimiento ordenado y priorizado, que puede ser ejecutado en diversos momentos del tiempo, adaptándose a las metodologías ágiles.

Así mismo, el modelo propuesto contiene las especificaciones necesarias del negocio, las relaciones y las restricciones, de manera que se puedan trazar las tareas, las pruebas y el modelo de navegación desde los requerimientos, para ser transformado por parte de la ingeniería guiada por modelos.

Es por tanto un modelo que cumple las condiciones de ambos paradigmas.

Esto permitirá, en un futuro, abordar proyectos basados en metodologías ágiles de una manera formalizada, al poder utilizar transformaciones automáticas para su desarrollo. Como trabajo futuro, restan tanto los métodos de transformación como el paso al tercer nivel, donde se realizarán una transformación a una plataforma tecnológica específica, así como la formalización del conjunto de técnicas ágiles sobre las que se soporta el modelo.

5 Agradecimientos

Esta investigación ha sido apoyada por el proyecto MeGUS (TIN2013-46928-C3-3-R) del Ministerio de Ciencia e Innovación, el proyecto Tempros (TIN2010-20057-C03-02), y el proyecto NDTQ-Framework (TIC-5789) de la Junta de Andalucía.

6 Referencias

1. Pressman, R.S. “What a Tangled Web We Weave”. IEEE Software, pp. 18-21. Jan.-Feb. 2000
2. Deshpande, Y. et al. “Web Engineering”. Journal of Web Engineering, Vol. 1, No.1, 003-017. 2002
3. Ambler, S.W. “Lessons in Agility from Internet-based Development.” IEEE Software, pp. 66-73. Mar-Apr, 2002
4. N. Koch. “Software Engineering for Adaptive Hypermedia Applications”. Ph. Thesis, FAST Reihe Softwaretechnik Vol (12), Uni-Druck Publishing Company, Munich. Germany. 2001
5. S. Ceri, P. Fraternali, P. Bongio. “Web Modelling Language (WebML): A Modelling Language for Designing Web Sites”. Conference WWW9/Computer Networks 33 (1-6) pp. 137-157. Mayo 2000

6. M.J. Escalona, G. Aragón. NDT. A Model-Driven approach for Web requirements. IEEE Transaction on Software Engineering. Vol. 34. Nº3. pp. 370-390. 2008.
7. Cohn, M. "Agile Estimating and Planning". NJ: Addison-Wesley. 2005.
8. Sutherland, J.; Schwaber, K. "The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game". 2011. http://www.scrum.org/Scrum-Guides. Accessed 2013.
9. Beck, K.; Andres, C. "Extreme Programming Explained: Embrace Change, Second Edition". Boston: Addison-Wesley. 2004.
10. Cockburn, A. "Crystal Clear: A Human-Powered Methodology for Small Teams". Boston: Addison- Wesley. 2004.
11. Poppendieck, M.; Poppendieck, T. "Lean Software Development. An Agile Toolkit". Boston: Addison-Wesley. 2003.
12. Palmer, RS "A Practical Guide to Feature-Driven Development". NJ: Prentice-Hall. 2002.
13. OMG. Unified Modeling Language: Superstructure, version 2.0. Specification, OMG, 2005. http://www.omg.org/cgibin/doc?formal/05-07-04.
14. Schwaber, K. "Controlled Chaos: Living on the Edge". American programmer. April 1996.
15. Ambler, S.W. "Lessons in Agility from Internet-based Development." IEEE Software, pp. 66-73. Mar-Apr, 2002
16. Ran, H. et al. 2008. "Agile Web Development with Web Framework". In proceedings of 4th International Conference on Wireless Communications, Networking and Mobile Computing. (Dalian, China, October 12 – 17 2008) WiCOM '08. IEEE
17. Armario, J.; Gutiérrez, J.J.; Alba, M.; García-García, J.A.; Vitorio, J.; Escalona, M.J., "Project Estimation with NDT". In proceedings of 7th International Conference on Software Paradigm Trends (Rome, Italy, July 24-27 2012). ICSOFT 2012: 120-126.
18. Rivero, JM; Grigera, J. ; Rossi, G; Luna, ER ; Koch, N "Towards Agile Model-Driven Web Engineering". In proceedings 23rd International Conference on Advanced Information Systems Engineering (London, ENGLAND, Jun 20-24 2011). CAiSE 2011.
19. Rivero, JM; Grigera, J.; Rossi, G; Luna; Montero, F; Gaedke, M. Information and Software Technology. Vol. 5, Issue 3, pp. 670-687, Jun, 2014.
20. Grigera G.; Matías-Rivero J.; Robles E.; Giacosa F.; Rossi G., "From Requirements to Web Applications in an Agile Model-Driven Approach" In Proceedings 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012.
21. Constantine L.; Lockwood L;. "Software for Use: A practical guide to the models and methods of usage-centered design" Reading, Mass.: Addison-Wesley, 1999.
22. Shepperd, M.; Schofield, C.; Kitchenham, B. "Effort Estimation Using Analogy". Dept. of Comput., Bournemouth Univ. Software Engineering, 1996.
23. Ton, H.: A Strategy for Balancing Business Value and Story Size. In: AGILE 2007 Conference (2007).
24. Torrecilla-Salinas, C.J.; Sedeño J.; Escalona M.J.; Mejías M. "Using Agile methods for infrastructure projects: A practical experience". 22nd International Conference on Information Systems Development (Seville, 2-4, Sep, 2013). ISD2013.
25. Torrecilla-Salinas, C.J.; Sedeño J.; Escalona M.J.; Mejías M. "Using an Agile framework to deliver e-Government services in public administrations". 21nd Annual Software Quality Management (London, 2-5, Sep, 2013). SQM2013.

Introducing Approximate Model Transformations

Javier Troya[1] and Antonio Vallecillo[2]

[1] Vienna University of Technology, Business Informatics Group, Austria
troya@big.tuwien.ac.at
[2] Universidad de Málaga, ETSI Informática, Spain
av@lcc.uma.es

Abstract. Model transformations dealing with very large models need to count on mechanisms and tools to be able to manage them. The usual approach to improve performance in these cases has focused on the use of concurrency and parallelization techniques, which aim at producing the correct output model(s). In this paper we present our initial approach to produce target models that are accurate enough to provide meaningful and useful results, in an efficient way, but without having to be fully correct. We introduce the concept of *Approximate Model Transformations*.

Keywords: Model Transformation, Approximation, Performance

1 Introduction

Model transformations are gaining acceptance as model-driven techniques are becoming commonplace. So far the community has mainly focused on the *correct* implementation of a model transformation, according to its specification [5,6,11], although there is an emergent need to consider other (non-functional) aspects such as performance, scalability, usability, maintainability and so forth. In particular, the study of the performance of model transformations is gaining interest as very large models living in the cloud need being transformed [1,2,9]. The usual approach to improve performance has focused on the use of concurrency and parallelization techniques.

In this paper we want to explore a different path. Our aim is to weaken the need to produce *correct* target models but *approximate* ones. Such approximate target models should be accurate enough to provide meaningful and useful results to users, but alleviate the need for the transformation to generate fully correct models—being able to produce such target models in much shorter time.

We call *Approximate Model Transformations* (AMTs) those model transformations that produce *approximate* target models. They are similar to the Approximation [10] or Randomized [7] Algorithms used in Computer Science and Operations Research.

This kind of model transformations are needed in various circumstances. The most obvious situation is when very large models have to be synthesized into much smaller models for decision making. We introduce a case study to present this problem. Let us consider a Wireless Sensor Network (WSN), where observation phenomena arrive and keep arriving as time moves forward. Consequently, the models grow with time (they are, in fact, *streaming* models [4]). Fig. 1 shows our metamodel for WSNs, developed with some ideas gathered from [8] and where four types of phenomena are

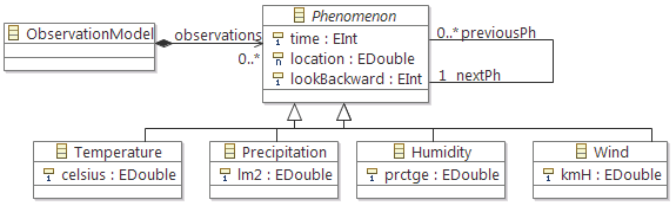


Fig. 1. Metamodel for observation phenomena in a WSN.

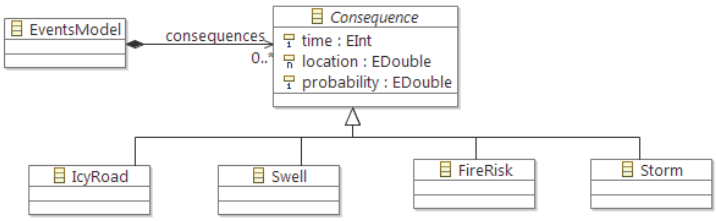


Fig. 2. Metamodel for defining consequences from observed phenomena.

identified. Each Phenomenon registers the time and location (in terms of the two terrestrial coordinates) where the observation takes place. Also, they keep record of the phenomena of the same type that was registered before and after the current one happened (previousPh and nextPh, respectively). As an inherent characteristic of streaming models, the latter relationship is not created when an object of type Phenomenon appears, but will be established later in time (when *future* phenomena appear). Reference previousPh points from a phenomenon to others phenomena of the same type that were obtained before, and only up to a certain point in time. Reference nextPh, in turn, points only to the next phenomena. Attribute lookBackward establishes the range of past observed phenomena – from the respective point in time of each phenomenon. Finally, each specialization of Phenomenon contains an attribute to keep the value obtained by the sensor node.

As explained in [8], to derive additional knowledge from semantically annotated sensor data, it is necessary to define and use a rule-based reasoning. In this way, when a group of sensor nodes provides information regarding for instance temperature and precipitation, then, using such rules, we can specify possible road conditions. One of these rules could be the following: if the temperature is less than 1 degree Celsius and it is raining, then the roads are potentially icy. To be able to express this kind of information in a model-based manner, we have created the metamodel shown in Fig. 2. It contains four different Consequences that are to be predicted according to the observed phenomena. Each consequence contains the time and location when and where it is predicted, as well as the probability that such consequence actually happens. Fig. 3 shows the preconditions in terms of observation phenomena, as well as the postconditions in terms of consequences, considered in our case study.

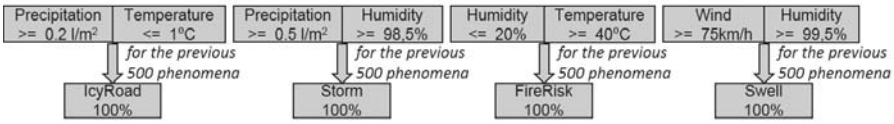


Fig. 3. Phenomena and Consequences.

In this paper, in the context of WSNs, we aim at reasoning over this kind of models not only considering the sliding window, i.e., the current information gathered from sensors that we have, but also taking into consideration that the information within such window can be potentially large. Thus, we focus on obtaining approximate predictions using AMTs by redefining some of the traditional operators used to traverse or query the models: allInstances(), select(), collect(), size(), etc. After this introduction, Sect. 2 presents our approach. Then, Sect. 3 describes an implementation and evaluation based on the case study and, finally, Sect. 4 draws some conclusions and future work.

2 Proposal

An AMT will likely not transform all the elements from the input model, for different reasons. For example, because they are not within the sliding window (e.g., they have already passed or have not arrived yet and there is no time to wait for them), or because the model is so large that we only work on a random sample of it [7]. This can result in some pending relationships in the output metamodel, what may make the output model not conform to its metamodel, and hence be *incorrect*. This kind of conformance is normally specified by means of constraints (also called contracts [3,5] in some contexts) that the model elements and their relations should respect.

In AMTs, it is also important the degree of *correctness* that the computed data have. In our case study, we consider that the probability of a consequence in a specific point in time is accurate according to certain rules as those shown in Fig. 3. Now, if the calculation of the probability is only based on a sample, and not on all the previous phenomena, the result may not be correct, just *accurate* enough for our purpose.

Approximate Operators. To elaborate our approach for defining AMTs, we apply a concept used in Randomized algorithms [7], namely *Random Sampling*. It is based on the idea that a small random sample from a population is representative of the population as a whole, and thus the properties of the sample can be used to determine some feature of the entire population.

Our proposal raises with the idea of redefining the common operators that current model transformation languages use to manage and operate with collections, such as allInstances(), collect(), select(), forAll(), etc. When transforming very large models, these operations become very expensive, performance-wise, because they have to traverse the whole model and deal with a large number of elements. If we reduce this number of elements, the transformation will execute faster. This is why we introduce a set of new collection operators, each one corresponding to an OCL collection operator.

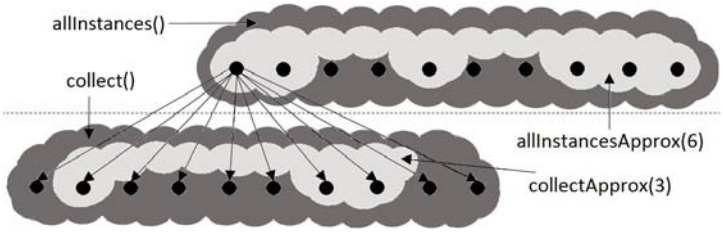


Fig. 4. Idea for Approximate Operators

The new ones end with "Approx" and incorporate an additional argument: an integer that represents the maximum number of instances the operator will process.

Fig. 4 shows this idea. In our case study, we may not need to obtain the predictions for all locations every time unit. Instead, we can randomly choose subsets of locations whose results can be extrapolated to surrounding areas, what corresponds to the upper part of the figure. Furthermore, since the creation of a consequence consists of the observation of a certain number of previous phenomena, we can also select a subset of them, what corresponds to the lower part of the figure. In this case, the values of the probabilities obtained in the consequences vary with respect to the originals, but they can be representative if they are close to the correct one.

3 Implementation and Evaluation

In our case study, the idea is to compute new consequences (conforming to the metamodel shown in Fig. 2) every time unit according to the phenomena observed by some sensor nodes (conforming to the metamodel shown in Fig. 1). This is potentially expensive, and the larger the number of sensor nodes deployed is, the longer it takes to process all data. We consider that the values of the measures gathered by two related sensor nodes (as shown in Fig. 3) has to be computed into a consequence if these nodes are not further than 10 away from each other, in Euclidean distance.

In the model we have considered, the slicing windows will contain 400 different locations, where the two locations which are collocated further away from each other have an Euclidean distance of 21.21. In fact, the sensor nodes in these locations have been randomly placed between coordinates $(0, 0)$ and $(15, 15)$. Sensor nodes of the four types in the metamodel (temperature, precipitation, humidity and wind) are distributed among these 400 locations, so that we have 100 points with each measure. Also, each Phenomenon keeps the history of the 500 previous phenomena of the same type. This means that attribute lookBackward has a value of 500 in our model.

We aim at obtaining the consequences for time 500. We have 200000 different phenomena in our input model, given how we have built it. Our transformation consists of one matched rule, four lazy rules and several helpers that perform the calculations. The lazy rules are called from the matched rule to build the EventsModel from an ObservationModel. Each lazy rule is used for obtaining consequences of the four different types, and each of them contains two inputs, as graphically depicted in Fig. 3. The way

Table 1. Accuracy for some consequences.

Location	Consequence	Prob. Original	Prob. AMT	Deviation
$(11, 10.2)$	Icy Road	78%	72.1%	5.9%
$(5, 10.8)$	Storm	47.2%	48%	0.8%
$(6.4, 3.2)$	Fire Risk	47.2%	49.4%	2.2%
$(8.1, 3.8)$	Swell	35%	28%	7%

we compute the consequences is merely a simplification of how it would be done in reality. Focusing for instance on icy roads, we say that the precipitation has to be equal or above $0.2l/m^2$ and the temperature equal or below 1 degrees Celsius, both during the previous 500 units of time, in order to have a 100% probability of icy roads. If not all the previous phenomena satisfy this condition, but some of them do, then we calculate the actual percentage by applying a simple cross-multiplication.

In the original transformation, a total of 28893 consequences are calculated in 111.2 seconds. An AMT has been constructed starting from the assumption that not all consequences have to be calculated every time unit, but calculating some of them is enough (upper part in Fig. 4). In the listing below we show how we make use of a selectApprox operation in order to take only 50 phenomena of each type (we show only the retrieval of Temperature phenomenon, it is the same for the others).

```
--Original Transformation
rule GenerateConsequences{
 from om : MM!ObservationModel
 using {
  ts : Sequence(MM!Temperature) = om.observations -> select (p |
           p.oclIsTypeOf(MM!Temperature) and p.time = 500);
(...)
--Approximate Model Transformation
(...)
  ts : Sequence(MM!Temperature) = om.observations -> selectApprox (50, p |
         p.oclIsTypeOf(MM!Temperature) and p.time = 500);
(...)
}
```

These operations are implemented now simply as helpers, although we plan to integrate them in the language as future work. Thus, since we choose a subset of the initial phenomena from which to obtain consequences, only 7362 are produced in the AMT, now in 22.1 seconds, what means a speedup of 5.03. We also realize an approximation according to the lower part of the figure. Now, instead of looking at the previous 500 phenomena of each type in order to calculate the consequence, we only consider half of the sample. As expected, the probability for the computed consequences is different than in the original transformation. Table 1 shows the values obtained in some locations in both approaches. As we can see, the deviation is not significant, so for this case study it is worth applying an AMT due to the gain in performance.

4 Conclusions and Future Work

This exploratory paper has presented a novel approach for the definition of Approximate Model Transformations (AMTs). We have explained its basic ideas and applied it in a case study. This paper presents just initial ideas that require deeper investigations at all levels. We were interested in exploring the possibility of defining AMTs, and our

initial results show that there are enough reasons to keep working on them. The work presented here does not pretend to be conclusive, or comprehensive, but to open the path for the modeling community to start working on it.

There are several open issues that we plan to address next. In the first place, we want to realize a comprehensive study on how accuracy can be measured depending on the particular scenarios, applications and model kinds. Second, we would like to provide formal and precise specifications for our approximate operators, and integrate them in the ATL language. Extending the approximate operators with a new argument that defines the maximum interval of time that we want the operator to execute is another line of work. Last, but not least, we need to study and develop methods for the appropriate design of AMTs. Although this will normally require a deep knowledge on the domain and the particular transformation scenarios, there is already a fair amount of work about the design of approximate and randomized algorithms that could be applicable in this context.

Acknowledgements. This work is partially funded by Research Project TIN2011-23795 and by the EC under ICT Policy Support Programme (grant no. 317859).

References

1. van Amstel, M., Bosems, S., Kurtev, I., Pires, L.F.: Performance in Model Transformations: Experiments with ATL and QVT. In: Proc. of International Conference on Model Transformations (ICMT). LNCS, vol. 6707, pp. 198–212. Springer (2011)
2. Burgueño, L., Troya, J., Wimmer, M., Vallecillo, A.: On the Concurrent Execution of Model Transformations with Linda. In: Proc. of the Workshop on Scalability in Model Driven Engineering (BigMDE 2013). ACM Digital Library (2013)
3. Cariou, E., Belloir, N., Barbier, F., Djemam, N.: OCL contracts for the verification of model transformations. Electronic Communications of the EASST 24 (2009)
4. Cuadrado, J.S., de Lara, J.: Streaming Model Transformations: Scenarios, Challenges and Initial Solutions. In: Proc. of International Conference on Model Transformation (ICMT 2013). LNCS, vol. 7909, pp. 1–16. Springer (2013)
5. Gogolla, M., Vallecillo, A.: *Tract*able Model Transformation Testing. In: Proceedings of the 7th European Conference on Modelling Foundations and Applications (ECMFA 2011). LNCS, vol. 6698, pp. 221–235. Springer (2011)
6. Guerra, E., de Lara, J., Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schönböck, J., Schwinger, W.: Automated verification of model transformations based on visual contracts. Autom. Softw. Eng. 20(1), 5–46 (2013)
7. Motwani, R., Raghavan, P.: Randomized algorithms. ACM Comput. Surv. 28(1), 33–37 (1996)
8. Sheth, A., Henson, C., Sahoo, S.S.: Semantic Sensor Web. IEEE Internet Computing 12(4), 78–83 (2008)
9. Tisi, M., Perez, S.M., Choura, H.: Parallel Execution of ATL Transformation Rules. In: Proc. of MODELS 2013. LNCS, vol. 8107, pp. 656–672. Springer (2013)
10. Vazirani, V.V.: Approximation Algorithms. Springer (2003)
11. Wimmer, M., Burgueño, L.: Testing M2T/T2M Transformations. In: Proc. of MODELS 2013. LNCS, vol. 8107, pp. 203–219. Springer (2013)

DSL-2-Browser: Un ejemplo de ejecución de un lenguaje específico del dominio en un navegador web

Álvaro Gutiérrez-Pérez, Luis-María García-Rodríguez, Rober Morales-Chaparro, and Fernando Sánchez-Figueroa

Quercus Software Engineering Group, Universidad de Extremadura
agutierrez|luismaex|robermorales|fernando@unex.es

Resumen. En este trabajo se expone, mediante un ejemplo, la viabilidad de ejecutar un DSL en un navegador web. Para ello se ha usado principalmente Xtext y GWT sobre el caso concreto de un DSL de visualización de datos. Aunque la propuesta se realiza a través de un ejemplo concreto, es posible su generalización para otros DSL.

Palabras clave: Desarrollo de software dirigido por modelos, Ingeniería Web, Lenguajes específicos del dominio, JavaScript

1. Introducción

Los lenguajes específicos del dominio (DSL) [1] surgen para acercar el desarrollo de software a los expertos del dominio. Paralelamente, la web se ha ido imponiendo como plataforma para la visualización de contenido y ejecución de código ante la creciente variedad de dispositivos con acceso a internet. Por todo ello, sería deseable llevar la potencia de los DSL a la omnipresencia de la web, es decir, poder describir mediante un DSL el código que se ejecuta en un navegador web.

En este documento se presenta un proceso para generar código JavaScript ejecutable en un navegador web a partir de un lenguaje específico de dominio. La propuesta se realiza a través de un ejemplo concreto, visualligence [2] un DSL para la visualización de datos. Este DSL permite al usuario crear su propia visualización de los datos de que dispone según la representación gráfica que desee, por tanto es un campo de aplicación que se puede beneficiar de la propuesta.

Partiendo de un determinado modelo en base a un DSL, mediante Xtext y la ayuda de una librería específica del dominio, se obtiene un modelo correspondiente en Java que puede ejecutarse correctamente en la máquina virtual como una aplicación independiente. Tras un proceso de adaptación de la librería, usando GWT se obtiene una descripción equivalente que puede ser ejecutada en un navegador usando únicamente tecnologías estándares. Respecto al ejemplo concreto que se va a detallar, actualmente el DSL se ejecuta convenientemente en la máquina virtual de Java, y la intención de este artículo es poder ejecutarlo en un navegador web mediante JavaScript. En la Fig. 1. se resumen estos pasos, que componen la propuesta presentada.

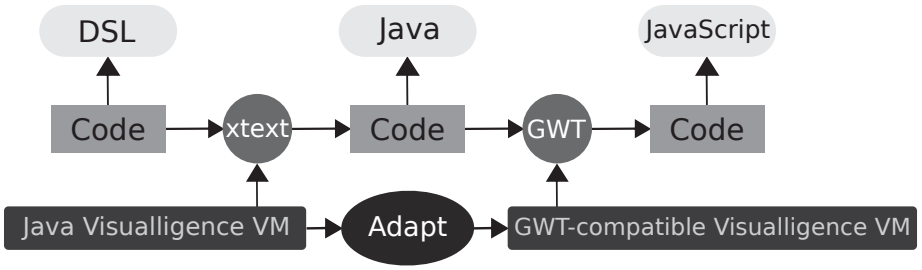


Figura 1. Vista general de la propuesta presentada.

El artículo se estructura de la siguiente forma: la Sección 2 presenta tecnologías existentes relacionadas con el problema a resolver. La Sección 3 presenta el caso de estudio: hilo conductor del desarrollo posterior, en la Sección 4. Por último, la Sección 5 contiene las conclusiones obtenidas junto con trabajos futuros que surgen a partir de nuestra propuesta.

2. Tecnologías y trabajos relacionados

A continuación se describen propuestas que se usan actualmente con un objetivo cercano al nuestro.

Xtext[1] es una plataforma para desarrollar DSLs basada en Eclipse. Usa una sintaxis extendida para la descripción de la gramática y genera automáticamente un modelo de clases de la gramática. Además, se puede lanzar un editor para escribir código basado en ese DSL.

Google Web Toolkit (GWT)[2] proporciona herramientas para el desarrollo de aplicaciones web. Permite desarrollar tanto el código del cliente como el del servidor en la misma tecnología (Java). Posteriormente, el código del cliente se "compila" a JavaScript y es ejecutado en el navegador.

CoffeeScript[3] es un lenguaje de programación que genera código JavaScript. Su principal atractivo es la simpleza comparado con JavaScript, permitiendo ahorrar gran cantidad de texto. **Source Maps**[4] es un estándar que permite relacionar unos ficheros de código con otros. Es usado para hacer corresponder código JavaScript ofuscado o generado con su correspondiente versión fuente. Así, a la hora de depurar, se muestra en el visor del navegador el código original en lugar de la versión que se está ejecutando.

asm.js[5] es un subconjunto de JavaScript muy optimizable, diseñado para ser muy eficiente. Existen compiladores de otros lenguajes a *asm.js* que permiten, por ejemplo, desarrollar un programa en C/C++ y ejecutar el código *asm.js*

[1] `http://www.eclipse.org/Xtext`
[2] `http://www.gwtproject.org/`
[3] `http://coffeescript.org/`
[4] `http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/`
[5] `http://asmjs.org/`

correspondiente en un navegador. **Native Client**[6] es un proyecto para permitir la ejecución de código nativo en un navegador dentro de un entorno controlado e independiente de la plataforma.

El reproductor **Flash** de Adobe permite reproducir contenido dinámico de forma homogénea en distintas plataformas. **WebCL**[7] propone un estándar para llevar la computación paralela al navegador, aprovechando los distintos núcleos y capacidades gráficas del hardware.

Estos proyectos aportan muchas soluciones, cada uno en su ámbito, pero se ven limitados en nuestro propósito de facilitar una comunicación más directa y formal con el cliente mediante un DSL. O bien proponen lenguajes específicos pero ligados a una tecnología concreta, que no permiten su uso directo en la web.

Como se ha visto, hay propuestas tanto para crear lenguajes específicos del dominio como para dotar a la web de nuevos lenguajes de programación. Sin embargo, en nuestro conocimiento, no existe ninguna que permita llevar a la web los lenguajes específicos del dominio, simplificando el desarrollo de código y facilitando la comunicación con el cliente. En la Tabla 1. hay un resumen de las tecnologías mencionadas y la evaluación de algunas características relevantes en nuestro trabajo.

Tabla 1. Características relevantes de las tecnologías similares a nuestra propuesta

	Ejecutable en el navegador	(Sin plugins)	Independiente de la gramática	Código fuente trazable
Xtext			✓	✓
GWT	✓	✓		✓
CoffeeScript	✓	✓		
Source Maps			✓	✓
asm.js	✓	✓		
Native Client	✓	✓	✓	
Flash	✓			✓
WebCL	✓	✓		✓
dsl-2-browser	✓	✓	✓	✓

Existen propuestas académicas [3] [4] relacionadas con DSLs y la web, que abordan problemas similares pero no el mismo.

3. Caso de estudio

La propuesta se presentará a través de un ejemplo, para ayudar a entender más fácilmente la utilidad de lo que se expone. El ejemplo que se usará consiste en un DSL diseñado para crear visualizaciones de datos para la web. Cada vez, más y más datos están disponibles en la Web, de fuentes cada vez más diversas.

[6] `https://developers.google.com/native-client/dev/`
[7] `http://www.khronos.org/webcl/`

En este contexto, visualligence [2] surge como una propuesta dirigida por modelos para desarrollar visualizaciones dirigidas por datos, multi-dispositivo, adaptativas, y elegidas según el perfil del usuario final. El núcleo de visualligence es un DSL pensado para usuarios no técnicos, con el que el dueño de los datos puede modificar, ajustar y desarrollar patrones visuales, así como intercambiarlos entre sí. Actualmente el DSL se ejecuta en la máquina virtual de Java, y la intención de este artículo es ejecutarlo en un navegador web mediante JavaScript.

El Listado 1. muestra un ejemplo de patrón de visualización descrito con el DSL de visualligence en el que se asocia un punto a cada dato.

```
1 loop dots for people { | person, i |
2     output <- Circle {
3         @x <- person.age
4         @y <- person.weight
5         @height <- 0.1
6         @width <- 0.1
7     }
8 }
```

Listado 1. Ejemplo de patrón visual que asocia un punto a cada dato

4. Propuesta

En esta sección se describe el procedimiento para partir de un DSL y obtener código ejecutable en un navegador. Primero se detallan una serie de pruebas que se han realizado para comprobar la viabilidad del proyecto, y posteriormente se aplica el mismo procedimiento a visualligence.

4.1. Experimentación

En este apartado se introduce el funcionamiento de GWT a través de las pruebas realizadas, mostrando los problemas que presentan y cómo se han resuelto.

Un proyecto de GWT consta de varias partes. Por un lado el código del cliente que es posteriormente compilado a JavaScript, y por otro el código del servidor. Además del código, el proyecto también necesita un fichero con extensión `.gwt.xml` que contiene información necesaria para el compilador de JavaScript, entre ella la clase principal o *punto de entrada* de la ejecución en el navegador. Para el propósito de esta propuesta el código del servidor no es relevante, así que nos centraremos en la parte del cliente, que es la que se ejecuta en el navegador.

Para las primeras pruebas se ha usado un DSL muy sencillo basado en un ejemplo de Xtext, con una gramática que contiene saludos. En el Listado 2. se puede ver la descripción en Xtext de esta gramática. La implementación se realiza creando un modelo de la máquina virtual (*jvmmodel*) para las clases

Java y usando el generador de Xtext para los ficheros adicionales de GWT. Se comprueba que este esquema funciona correctamente generando un editor de Xtext para esa gramática, escribiendo un código de ejemplo y observando que la generación es correcta y se visualiza el resultado en el navegador.

```
1 PackageDeclaration:
2   'package' name=QualifiedName model=Model;
3 Model:
4   'Greetings' 'from' name=ID ':'? greetings+=Greeting*;
5 Greeting:
6   'Hello' name=ID surnames+=ID* '!'?;
```

Listado 2. Gramática de ejemplo usada durante las primeras pruebas.

El siguiente paso es poner a prueba los límites de GWT. Primero se hace incluyendo librerías externas precompiladas, en forma de archivo *jar*. Sin embargo, se comprueba que GWT necesita el código fuente de todo el código que se incluye, puesto que la transformación se realiza directamente sobre éste.

A continuación, se incluye en el proyecto el código fuente de una librería externa de cierto tamaño con el fin de determinar el nivel de soporte de GWT de la librería estándar de Java, y se intenta compilar. Dado que hay clases no soportadas por el compilador de GWT[8], al encontrarlas genera un mensaje de error y se detiene el proceso sin llegar a generar código.

Estas importantes limitaciones de GWT se deben tener en cuenta posteriormente a la hora de diseñar la librería del DSL que permita su ejecución en el navegador. Es decir, se debe disponer del código fuente de todas las librerías externas que se vayan a incluir y éstas deben usar únicamente las clases soportadas para poder usarlas sin problemas.

4.2. Visualligence

Una vez determinadas las posibilidades y limitaciones de GWT, habiendo constatado que es posible llevar a cabo la transformación, se procede a trabajar sobre el código de un proyecto real, visualligence. Se debe realizar una adaptación del código actual a uno compatible con GWT que cumpla sus limitaciones.

El principal problema se encuentra en las clases del paquete de entrada y salida `java.io`, dado que la mayor parte de ellas carecen de soporte en GWT. Esto limita enormemente la capacidad para trabajar con flujos de entrada y salida, un pilar fundamental en visualligence. Por tanto, se ha desarrollado una jerarquía de entrada y salida paralela a la proporcionada por la librería estándar de Java, pero sin usar referencias a ellas. También tuvieron que sustituirse o reemplazarse por implementaciones propias otras clases no soportadas por GWT, entre ellas `Scanner`, del paquete `java.util`.

[8] `http://www.gwtproject.org/doc/latest/RefJreEmulation.html`

Una vez resueltos estos problemas, se realiza la compilación a JavaScript sin problemas, y el código generado es ejecutado satisfactoriamente en el navegador.

En la Fig. 2. se puede ver el resultado: en el recuadro de la izquierda están los datos de entrada que recibe la aplicación. En el recuadro central se muestra la salida que produce la ejecución del patrón mostrado en el Listado 1. con los datos de entrada anteriores. Por último, en el recuadro de la derecha se muestra el resultado final tras ser interpretada la salida anterior por el navegador.

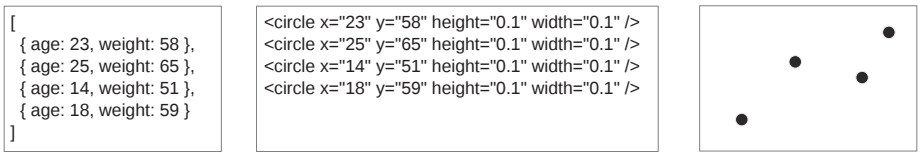


Figura 2. Datos de entrada, salida de visualligence, visualización en el navegador.

5. Conclusiones y trabajos futuros

Esta propuesta se ha descrito a través de un ejemplo realizado con una tecnología concreta. Esto podría dificultar la generalización del proceso para otras tecnologías. Sin embargo, esto no es problema, puesto que para otras tecnologías existen otras herramientas (como se ha visto en los antecedentes) que podrían usarse de forma similar a la descrita aquí.

En el futuro usaremos el mismo procedimiento con otro DSL, comprobando hasta qué punto el mismo mecanismo es reutilizable para distintos lenguajes. También sería necesario realizar una comparativa de rendimiento usando un ejemplo concreto en el DSL del caso de estudio, comparando una ejecución nativa con su correspondiente en el navegador. Por último, automatizar los distintos procedimientos, que ahora se ejecutan manualmente, producirá una transformación más sencilla y generalizable, posibilitando su uso en otros ámbitos.

Referencias

1. Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.
2. Rober Morales-Chaparro, Juan Carlos Preciado, and Fernando Sánchez-Figueroa. Visualizing search results: Engineering visual patterns development for the web. In Stefano Ceri and Marco Brambilla, editors, *Search Computing*, volume 7538 of *Lecture Notes in Computer Science*, pages 127–142. Springer Berlin Heidelberg, 2012.
3. Juan José Cadavid, Juan Bernardo Quintero, David Esteban Lopez, and Jesus Andrés Hincapié. A domain specific language to generate web applications. In *CIbSE*, pages 139–144, 2009.
4. Eelco Visser. Webdsl: A case study in domain-specific language engineering. In *Generative and Transformational Techniques in Software Engineering II*, pages 291–373. Springer, 2008.

MDD vs. traditional software development: A practitioner's subjective perspective[1]

Yulkeidi Martínez[1], Cristina Cachero[2] and Santiago Meliá[2]

[1]Universidad Máximo Gómez Báez de Ciego de Avila, Cuba
yulkeidi@gmail.com
[2]Universidad de Alicante, Spain
{ccachero, santi}@dlsi.ua.es

Abstract. Today practitioners have a myriad of methods from which to choose for the development of software applications. However they lack empirical data that characterize these methods in terms of usefulness, ease of use or compatibility, all of them relevant variables to assess the developer's intention to use them. In this context, we propose to compare three methods, each following a different paradigm (Model-Driven, Model-Based and the traditional, code-centric, respectively) with respect to its intention to use by junior software developers while developing the business layer of a Web 2.0 application. To do that, we have conducted an experiment with graduate students of the University of Alicante. The application developed was a Social Network, which was organized in three different modules. Subjects were asked to use a different method for each one of the three modules, and then answer a questionnaire that gathered their perceptions during its use. The results show that the method that followed the Model-Driven development paradigm is regarded as the most useful, although it is also regarded as the more difficult to use. They also show that junior software developers feel comfortable with the use of models, and are likely to use them if accompanied by a model-driven development environment.

Keywords: MDD, MBD, code-centric development, experiment, usefulness, ease of use, compatibility, intention to use.

1 Context of the proposal

It is a well-known fact the Software Engineering (SE) community advocates the use of models in order to improve software development practices. However, up to now, the quality and quantity of the empirical analyses that demonstrate the true impact of these modeling techniques over the final developer's satisfaction are still very low.

[1] Yulkeidi Martínez, Cristina Cachero, Santiago Meliá: MDD vs. traditional software development: A practitioner's subjective perspective. Information & Software Technology 55(2): 189-200 (2013)

In this sense, Fowler [1] describes three different modes in which modeling languages (and UML in particular) can be used: sketch, blueprint and programming language. Starting with code-centric approaches (e.g. Scrum, XP, etc.), they usually promote the use of, at most, whiteboard sketching that permits to communicate the main ideas among different team members in a quick way. Also, the implementation is usually tackled manually. When the Model-Based paradigm is followed, its model discipline generates a blueprint, usually with the help of a UML tool. These UML tools usually only permit to obtain a partial implementation (usually plain classes with attributes and empty methods). Finally, the Model-Driven paradigm typically relies on fully-fledged models that are usually supported by a Domain-Specific Language (DSL). This paradigm requires modeling to be carried out by means of a Model-Driven CASE tool that permits to generate, either partially (semiautomatic) or in full (automatic), the final implementation. Moreover, there is the underlying assumption that using development methods that rely on models and tools with code generation capabilities improves the overall experience of the developer for when he/she is developing applications.

However, we agree with [3] that the low level of adoption of MDD approaches may be partly due to the fact that method assessment efforts still revolve mostly around method technological features (such as separation of concerns, the availability of tools or artifact traceability, to name just a few) while paying little attention to the developers' attitudes and perceptions of the method. It is therefore of paramount importance to the discipline to go into greater depth in the study of such perceptions and the variables that actually impact them (beyond the primary characteristics of the method). To advance on this path, in this paper we present an empirical study that studies the variables that may impact the actual use of methods (code-centric, MBD and MDD). In contrast to other studies, the intra-subject design used in this study favors the detection of perceived advantages-disadvantages of each approach with respect to the others.

Another important contribution of this work is that proposes a tailored theoretical method adoption model that can be used as a starting point to study the adoption possibilities of existing or new development methods, based as it is on well-known evidence from different fields. Such a theoretical framework points to perceived ease of use, perceived usefulness, compatibility as being the main variables that influence the intention to adopt a given method.

2 Description of the experiment

During the months of January and February of 2011, a controlled quasi-experiment following the GQM template was conducted at the University of Alicante. The goal of this experiment was to evaluate the perceive usefulness, ease of use and compatibility of three methods after subjects develop the business layer of a Social Network using three different approaches: code-centric (based on C# and the .NET framework), MBD (based on UML and supported by the Rational Software Modeler tool) and MDD (based on the OOH4RIA approach [2] and supported by the OIDE tool).

The subjects were 26 students of the Master in Web Software Development at the University of Alicante that had received training in all three methods.

This study was designed taking into account the common belief that the use of models improves the developers' perceived usefulness of the method, although it somehow made the development process more difficult. Contrary to our assumptions and the related work, using models in the context of an MBD environment is regarded as useless and difficult, even if the subjects did not show a great degree of incompatibility with the approach. On the contrary, the MDD method was regarded as the least compatible with developers' current practices, but the most useful in the long run. The fact that four out of five teams chose to continue using the MDD method suggests that usefulness seems to have a much greater impact on intention to use a method than compatibility, at least among junior developers.

References

1. M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 2004.
2. Meliá, S., Gómez, J., Pérez, S., Diaz, O. A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA. Eighth ICWE, Yorktown Heights, USA, 2008.
3. Walderhaug S., Mikalsen M., Benc I., Erlend S., Factors affecting developers' use of MDSD in the Healthcare Domain: Evaluation from the MPOWER Project, in: From Code Centric to Model Centric Software Engineering: Practices, Implications and ROI. Workshop at ECMDA, 2008.

Datos de publicación en revista de impacto

Título: MDD vs. traditional software development: A practitioner's subjective perspective

Autores: Yulkeidi Martínez, Cristina Cachero and Santiago Meliá

Revista: Information and Software Technology,

Volumen: 55, **Número:** 2**,** Febrero 2013

Páginas: 189-200

Año: 2013

Editorial: Elsevier

ISSN: 0950-5849

DOI: http://dx.doi.org/10.1016/j.infsof.2012.07.004

Indicios de Calidad

Revista: Information and Software Technology

Journal Citation Index:

Factor de impacto (2012): 1,522

Factor de impacto en los últimos 5 años (2012): 1,692

Categoría: Computer Science, Software Engineering

Tercil: Primero

Posición : 23 de 105

Nº citas: 7

PRISMA: Model-Driven Development of Aspect-Oriented Software Architectures*

Jennifer Pérez[1], Isidro Ramos[2], Jose A. Carsí[2], Cristóbal Costa-Soria[3]

Technical University of Madrid (UPM) - ETSISI-CITSEM, Madrid, Spain
jenifer.perez@eui.upm.es
Universidad Politécnica de Valencia (UPV) - DSIC, Valencia, Spain
iramos@dsic.upv.es, pcarsi@dsic.upv.es
Global Metanoia S.L., Paterna Technological Science Park (Valencia), Spain
ccosta@globalmetanoia.com

Abstract. This summary presents a methodology for supporting the development of AOSAs following the MDD paradigm. This new methodology is called PRISMA and allows the code generation from models which specify functional and non-functional requirements.

Keywords: MDD, Aspect-Oriented Software Architectures, code generation

1. Summary

Currently, mature MDD methodologies are required for supporting the code generation from models that specify non-functional requirements. Aspect-Oriented Software Architectures (AOSA) emerged to deal with the design of both, functional requirements and non-functional requirements, which opened an important challenge in the software engineering field: the definition of a methodology for supporting the development of AOSAs following the MDD paradigm. This new methodology should allow the code generation from models which specify functional and non-functional requirements. This contribution presents an approach, called PRISMA, which deals with this challenge. PRISMA follows the MDD approach by enabling software architects to define AOSA models, which allow the generation of the final code of AOSAs. The tasks of the software architect are facilitated thanks to the fact that: (i) the level of abstraction provided by models is higher than the provided by programming languages, and (ii) the code is automatically generated from models. The PRISMA MDD process is based on MOF and its models undergo the different levels of refinement during the process. Therefore, the PRISMA MDD process is divided into two main steps: (i) from the PRISMA Metamodel to PRISMA Type Models, and (ii) from PRISMA Type Models to PRISMA Configuration Models (see Fig. 1)

The PRISMA metamodel defines the PRISMA model and establishes its properties in a precise way (see 1, Fig. 1). To use these primitives in a modeling context, the PRISMA CASE tool provides the PRISMA metamodel through a graphical language to model PRISMA software architectures (see 2, Fig.1). PRISMA type models are

*Published in: Jennifer Pérez, Isidro Ramos, José A. Carsí, Cristóbal Costa-Soria: Model-Driven Development of Aspect-Oriented Software Architectures. Journal of Universal Computer Science, vol. 19, no. 10 (2013), 1433-1473, accepted: 27/5/13 © J.UCS. DOI: 10.3217/jucs-019-10-1433. Rankings: JCR: 0.762 , Journals CORE: A, Journals Microsoft Academic Research/Computer Science: 316 position of 1363

described using the concepts that are defined in the metamodel, guaranteeing that every PRISMA type model is defined conforming to the PRISMA metamodel (see 4, Fig. 1). The PRISMA MDD process assists the architect by providing mechanisms for the verification of models: this allows the detection of modelling mistakes, and keeps them from spreading throughout the rest of the stages. This verification distinguishes two kinds: verification rules that must always be satisfied (hard constraints), and verification rules that must be satisfied once the model has been completely finished (weak constraints). Once the PRISMA type model has been completely described, the architect can proceed to generate both the PRISMA AOADL specifications and the C# code corresponding to this model (see 4, Fig.1).In addition, since a PRISMA type model is a generic system architecture (banking system, tele-operated robot, etc.), the model can be reused for different specific systems. Once its code has been generated, the architect can proceed to define a specific system model, called configuration model. Every configuration model conforms to a PRISMA type model thanks to the fact that PRISMA configurations are defined using the concepts that have been previously defined in its PRISMA type model as modelling primitives. They are provided by PRISMA CASE, which automatically generates a domain-specific graphical modelling tool to configure specific software architectures from the type models, and generating the C# application code corresponding to the configuration model (see 5 and 6, Fig. 1).

The PRISMA MDD process is presented as an important advance in the automatic generation of code from AOSAs thanks to its automatic generation of AOADL specification, C# code and a domain-specific model to configure the final PRISMA software architecture of a system. This domain-specific model reduces the gap between the user and his/her knowledge about modelling. In addition, it has been applied to different kind of applications to validate its code generation capability: the robots TeachMover and Agrobot. The results revealed a high percentage of automatic code generation from PRISMA AOSA models: 94,3% for the TeachMover and 68,1% for the Agrobot, which correspond with 5686 and 1125 Generated Lines of Code. These code generation results reveal that the complete automatic code generation from AOSA models is feasible.

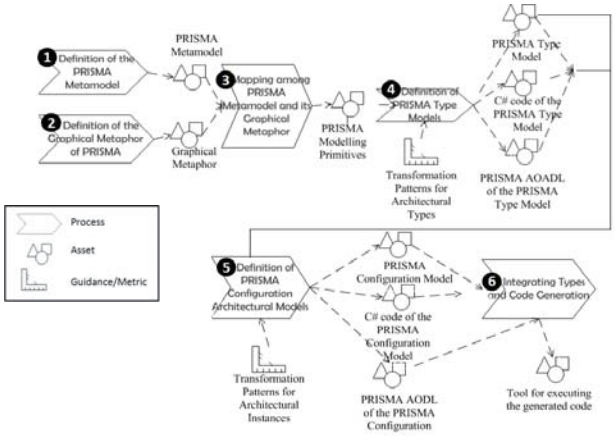


Fig. 1. MDD Process from the PRISMA Metamodel to Applications

A Language for End-user Web Augmentation: Caring for Producers and Consumers Alike*

Oscar Díaz, Cristóbal Arellano, Maider Azanza

University of the Basque Country (UPV/EHU), San Sebastián (Spain)
(oscar.diaz, cristobal.arellano, maider.azanza)@ehu.es

Abstract. This work advocates for the use of Domain Specific Languages to empower hobby programmers to achieve their own scripts for Web Augmentation (a special kind of applications to customize web pages). The work follows Mernik's methodology whereby the DSL is gradually constructed from the feature model to the concrete syntax.

Keywords: Web Augmentation, Domain Specific Language, JavaScript

1 Problem Statement

Web Augmentation (WA) superimposes additional content/layout/navigation over existing Web applications at the time the page is loaded. Examples of what this technology generically enables include reorganizing page content, supplementing page data, etc. Concrete examples can be found at *userscripts.org*, a popular repository for augmentation scripts. This site reports thousands of consumptions (which evidences the effectiveness of the approach) but a far smaller number of providers. The use of general-programming languages such as *JavaScript* partially explains why Web Augmentation production is so far limited to programmers. These scripts are prone to malware, and difficult to create and maintain. As a result, only dedicated programmers are prepared to *produce* scripts, and only courageous *consumers* are willing to install them. We strive to depart from this situation, moving WA to a scenario where end users *prosume* scripts as easily as they currently *prosume* pictures or videos.

2 Contribution

We propose Domain Specific Languages (DSLs) as a promising way to fulfil the previous requirements. From a *producer perspective*, DSLs facilitate development (e.g. domain abstractions are closer to how users conceive the problem, facilitating engagement and production of scripts), debugging (e.g. tracing is also

* Oscar Díaz, Cristóbal Arellano, Maider Azanza: **A Language for End-user Web Augmentation: Caring for Producers and Consumers Alike**. ACM Transactions on the Web 7(2): 9 (2013) http://dx.doi.org/10.1145/2460383.2460388

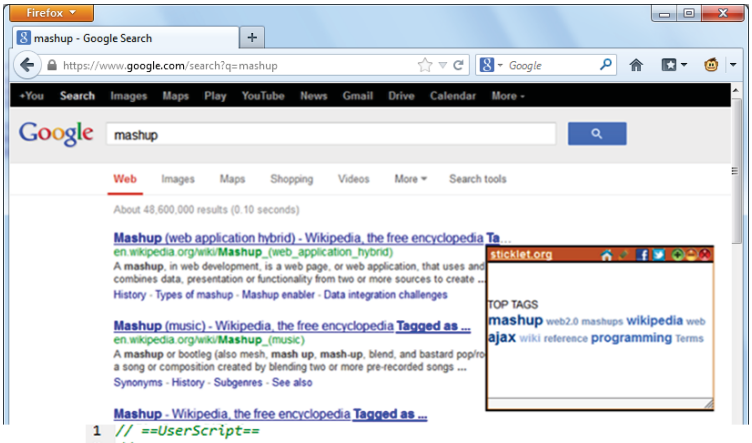


Fig. 1. *Sticklet* for augmenting *Google*-retrieved URLs with *Delicious* tags. Button *"Tagged as..."* is added. On clicking, *Delicious'* tags are inserted. In the code (line 14), *http://previous.delicious.com/url?url=$link* retrieves tags related to the URL kept in *$link*.

conducted in DSL terms) and maintenance (e.g. abstraction brings declarativeness). Target producers are hobby programmers. They should know about URLs, and a bit of *HTML* is recommended though not strictly necessary. They do not need to know about either *JavaScript* or any other programming language as the proposed DSL does not have explicit control instructions (i.e. neither 'while' nor 'if'). From a *consumer perspective*, DSLs bring trust due to both code understanding (DSL declarativeness) and built-in security (coded in the DSL engine). In this way, consumers only need to rely on the provider of the DSL engine (who can be externally certified e.g. as a *Mozilla* add-on partner) rather than trusting a potentially unknown *JavaScript* programmer.

Based on the previous observations, we implemented *Sticklet*, a DSL for Web Augmentation. Figure 1 provides an example of how an expression in the *Sticklet* DSL looks like, and the impact on the target page. The main contribution of this work rests on considering not only producer but also consumer concerns on handling *Sticklet* expressions. The DSL engine is available for public download as a *Firefox* add-on at *https://addons.mozilla.org/firefox/addon/Sticklet/*.

Herramienta de soporte en procesos de modernización, para las fases de ingeniería inversa y reestructuración*

Víctor M. Pavón, Roberto Rodríguez-Echeverría, Fernando Macías, Pedro J. Clemente, and Fernando Sánchez-Figueroa

Universidad de Extremadura,
Quercus Software Engineering Group
{victorpavon, rre, fernandomacias, pjclemente, fernando}@unex.es
http://quercusseg.unex.es

Resumen. El proyecto MIGRARIA define un proceso dirigido por modelos para la modernización de aplicaciones web heredadas. Una vez definido y validado el proceso, se ha puesto en marcha el desarrollo de una herramienta que asista al ingeniero de modernización en las fases de ingeniería inversa y reestructuración. El objetivo de este trabajo es presentar el prototipo de esta herramienta, cuyas funcionalidades base son: ofrecer un editor de modelos MIGRARIA MVC que facilite la comprensión del sistema heredado; facilitar la navegación bidireccional entre los artefactos software de la aplicación heredada y su representación abstracta (modelos MIGRARIA MVC); y simplificar la configuración y automatizar la ejecución del proceso de ingeniería inversa.

Palabras clave: Ingeniería Inversa, Desarrollo de Software Dirigido por Modelos, MDE, RIA.

1 Introducción

Los procesos de modernización se dividen en tres fases fundamentales: ingeniería inversa, reestructuración e ingeniería directa. La primera fase de ingeniería inversa consiste en el análisis de la aplicación heredada para obtener una representación abstracta de la misma. El objetivo principal del proyecto MIGRARIA ha consistido en la definición de un proceso sistemático, semiautomático y reutilizable para la modernización Aplicaciones Web Heredadas (AWH) a Rich Internet Applications (RIA) mediante métodos y herramientas propias del Desarrollo de Software Dirigido por Modelos. El proyecto MIGRARIA arranca en el año 2011 y en este tiempo se han desarrollado diferentes líneas de trabajo [1,2]. La principal ha sido la implementación del proceso de ingeniería inversa para AWH desarrolladas utilizando el framework Struts. En la parte superior de la Fig. 1 se

* Trabajo financiado por el Ministerio de Ciencia e Innovación, Proyecto TIN2011-27340; la Consejería de Empleo, Empresa e Innovación del Gobierno de Extremadura, Ayuda GR10129; y el Fondo Europeo para el Desarrollo Regional (FEDER)

muestran los pasos seguidos en el proceso de ingeniería inversa. En la primera fase se hace uso de la herramienta MoDisco[1] para generar una serie de modelos dependientes de la tecnología a partir de los artefactos software de la AWH (ficheros Java, JSP y XML). Estos modelos se utilizan como entrada de una cadena de transformaciones de la que se obtiene la representación abstracta de la AWH. En este trabajo se presenta el desarrollo de una herramienta[2] para asistir al ingeniero de modernización en las fases de ingeniería inversa y reestructuración, utilizando la representación conceptual de la AWH. Las partes implementadas para este trabajo se encuentran destacadas en la Fig. 1.

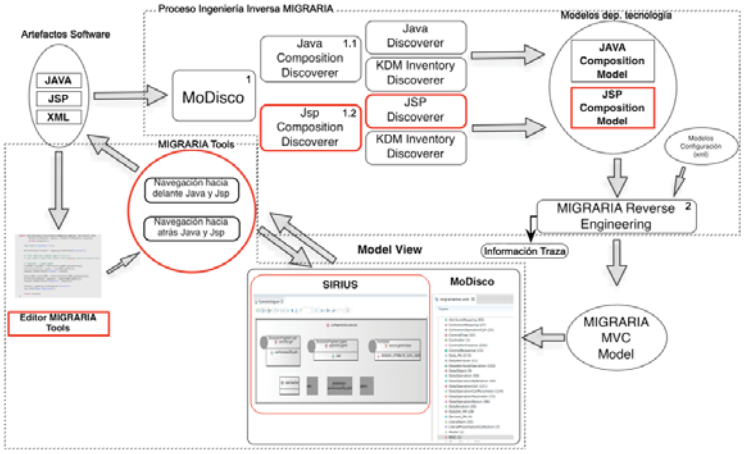


Figura 1: Proceso de ingeniería inversa MIGRARIA

2 Descripción de la herramienta

El primero de los objetivos es la definición de un editor gráfico para el modelo MIGRARIA MVC que facilite la comprensión del sistema heredado. Dentro del proyecto MIGRARIA, se ha definido una sintaxis concreta para el metamodelo MIGRARIA MVC mediante dos notaciones gráficas (una para la especificación del modelo y otra para la especificación conjunta de la vista y el controlador) y una notación textual (para la especificación del controlador), esta última todavía se encuentra en desarrollo. El editor para las notaciones gráficas se ha implementado mediante el uso de la herramienta Sirius[3], especialmente diseñada para la definición de editores gráficos. Se ha seleccionado Sirius por dos motivos principales: es la base de Ecore Tool a partir de la versión 2, y por otro lado mantiene de forma independiente la representación gráfica del modelo y el contenido.

El segundo de los objetivos se centra en facilitar la navegación bidireccional entre los artefactos software de la aplicación heredada y la representación abs-

[1] `http://www.eclipse.org/MoDisco/`
[2] Vídeo de demostración de la herramienta - http://goo.gl/doo9Nw
[3] `http://www.eclipse.org/sirius/`

tracta (modelos MIGRARIA MVC). Ésta es una tarea bastante frecuente en el proceso de verificación del modelo y comprensión de la representación generada por parte del ingeniero de modernización. Esta tarea se puede organizar en dos escenarios fundamentales. El primero consistiría en la navegación hacia atrás (o hacia abajo): desde la representación abstracta al artefacto software origen. El segundo escenario consistiría en la navegación hacia delante (o hacia arriba): desde un elemento de los artefactos software a su representación abstracta. Para llevar a cabo estas actividades es necesario que el ingeniero de modernización posea un conocimiento muy detallado de los artefactos software origen, el metamodelo destino y de las transformaciones definidas entre ambos.

En el escenario de navegación hacia atrás, se parte desde la selección de un elemento en el editor gráfico de modelos MIGRARIA MVC implementado basado en Sirius para terminar en un editor de texto con el código del artefacto software que genera el elemento concreto seleccionado. Este proceso se implementa mediante dos pasos. Un primer paso de navegación desde el modelo independiente de la tecnología MIGRARIA MVC al modelo dependiente de la tecnología, y un segundo desde el modelo dependiente al artefacto software correspondiente.

El primer paso utiliza la información de traza generada por las reglas de transformación modelo a modelo (*Model-to-Model Transformation,* MMT) del proceso de reingeniería de MIGRARIA, que se han implementado en ATL. Las alternativas evaluadas, para la generación de la información de traza, fueron: (1) añadir de forma manual en las reglas de transformación la generación del modelo de traza; (2) la utilización de la herramienta iTrace[4] que genera el código para la trazabilidad de forma automática; (3) utilización de la máquina virtual para ATL EMFTVM[5] que genera un modelo de traza sin tener que añadir código específico para ello en la transformación. Ésta última ha sido la alternativa seleccionada. Partiendo del elemento seleccionado en el editor de Sirius, la navegación se implementa consultando en el modelo de traza cuál o cuáles son los elementos origen en los modelos dependientes de la tecnología del elemento MIGRARIA MVC seleccionado.

El segundo paso de la navegación realiza la navegación desde un elemento del modelo dependiente de la tecnología al código fuente correspondiente. Se hace uso de la funcionalidad de trazabilidad y navegación entre código fuente y modelo dependiente de la tecnología que MoDisco ofrece en su discoverer Java Inventory Model. Este discoverer además de generar el modelo Java de la aplicación genera un modelo de inventario en el que almacena la región que ocupa cada elemento del modelo Java en el fichero de código en el que se define. Haciendo uso de este modelo de inventario, MoDisco implementa la navegación entre modelo dependiente de la tecnología y los artefactos software de un proyecto Java. Esta navegación permite que desde el browser de MoDisco se abra el fichero Java y se seleccione el código que se corresponde a un elemento del modelo de forma automática. Esta misma funcionalidad se ha extendido para soportar artefactos software de otro tipo, como JSP o XML.

[4] `http://www.kybele.etsii.urjc.es/itracetool/`
[5] `https://wiki.eclipse.org/ATL/EMFTVM`

El escenario de navegación hacia delante, por su parte, ha supuesto la implementación de un editor propio para código Java que extiende el ofrecido por Eclipse. Este editor permite seleccionar el tipo de elemento del modelo MIGRARIA MVC que nos interesa. A partir del tipo de elemento seleccionado y haciendo uso de la información de navegación hacia atrás se obtiene la lista de regiones del código Java que se han identificado como origen de los elementos del tipo seleccionado. Esta información es utilizada para resaltar en el editor el código que está asociado con la generación de elementos del tipo seleccionado y, además, se almacena para facilitar la navegación hacia delante. Una vez dentro del editor con las regiones resaltadas, al abrir el menú contextual de Eclipse sobre una región se puede navegar hacia delante, mostrando el elemento correspondiente en el editor de Sirius. Esta navegación se implementa también en dos pasos: (1) a partir de la zona del editor en la que se ha activado el evento de navegación obtener la región correspondiente y con ello el elemento del modelo dependiente de la tecnología con el que se relaciona; y (2) navegar desde el elemento dependiente de la tecnología al elemento del modelo MIGRARIA MVC. El primer paso básicamente consiste en identificar dentro de qué región estamos invocando el evento de navegación. Al obtener la región también se obtiene el elemento dependiente de la tecnología ya que junto con la regiones se almacena esta información. El segundo paso de navegación, del elemento dependiente de la tecnología al elemento de MIGRARIA MVC, realiza la búsqueda en sentido contrario al primer paso de la navegación hacia atrás, por lo que se ha implementado siguiendo la misma aproximación.

3 Conclusiones y trabajo futuro

Las características desarrolladas hasta el momento y presentadas en este trabajo han permitido reducir de forma considerable el esfuerzo de los ingenieros de modernización. La primera de las líneas de trabajo futuro es la definición de una herramienta que permita la adaptación de la cadena de transformación a nuevos casos de estudio. La segunda de las líneas es la automatización de la ejecución del proceso de reingeniería para reducir al mínimo la intervención del ingeniero de modernización.

Agradecimientos. Este trabajo ha sido parcialmente desarrollado por Enrique Rodríguez González como parte de su Trabajo Fin de Carrera.

Referencias

1. Roberto Rodríguez-Echeverría. *Un Proceso de Modernización de Aplicaciones Web a RIA*. PhD thesis, University of Extremadura, 2014.
2. José María Conejero, Roberto Rodríguez-Echeverría, Fernando Sánchez-Figueroa, Marino Linaje Trigueros, Juan Carlos Preciado, and Pedro J. Clemente. Re-engineering legacy web applications into rias by aligning modernization requirements, patterns and ria features. *Journal of Systems and Software*, 86(12):2981–2994, 2013.

CERVANTES: Un framework para el diseño y desarrollo de sistemas distribuidos

M.A. Barcelona[1,2], L. García-Borgoñón[1,2], J.I. Calvo[1], and M.J. Escalona[2]

[1] Instituto Tecnológico de Aragón, c/ María de Luna 7, 50018 Zaragoza,
mabarcelona@ita.es, laurag@ita.es, jicalvo@ita.es
[2] Universidad de Sevilla, Av. Reina Mercedes S/N, 41012 Sevilla,
miguel.barcelona@iwt2.org, laura.garcia@iwt2.org, mjescalona@us.es

Resumen En la actualidad vivimos una continua evolución en los sistemas informáticos que nos rodean: cada vez son de mayor tamaño, cubren más funcionalidades y es necesario que interactúen con otros sistemas ya existentes. El desarrollo de estos sistemas es por tanto una tarea compleja que sin embargo ha evolucionado a un menor ritmo. En los últimos años han surgido iniciativas como el desarrollo dirigido por modelos (*MDE*), la computación basada en servicios (*SOC*) o la ingeniería del software orientada a servicios (*SOSE*). En este trabajo se presenta una herramienta, basada en los conceptos de *MDE* y *SOC*, que desde el año 2002 se ha venido utilizando en la práctica con la industria para el diseño y desarrollo de sistemas distribuidos.

Palabras Clave *Model Driven Engineering*, *Service Oriented Computing*, *Service-Oriented System Engineering*

1 Motivación

En la actualidad vivimos una continua evolución en los sistemas informáticos que nos rodean: cada vez son de mayor tamaño, cubren más funcionalidades y es necesario que interactúen con otros sistemas ya existentes. El desarrollo de estos sistemas es por tanto una tarea compleja que sin embargo ha evolucionado a un menor ritmo. Para gestionar esta complejidad en los últimos años han surgido nuevos paradigmas e iniciativas como el desarrollo dirigido por modelos (*Model Driven Engineering, MDE*)[1], la computación basada en servicios (*Service-Oriented Computing, SOC*)[2] o la ingeniería del software orientada a servicios (*Service-Oriented System Engineering, SOSE*)[3]. Enmarcado en este complejo contexto, en el año 2001, antes de que surgiera un creciente interés por *MDE* y *SOC*, nos planteamos crear un marco que cubriera las siguientes necesidades:

- Debía servir para crear sistemas de sistemas en un entorno heterogéneo y distribuido.
- Se debía poder especificar un sistema de forma abstracta, sin entrar en los detalles de implementación.

- Debía facilitar la monitorización de los diversos subsistemas dentro del entorno distribuido en el que opera.
- Debía permitir describir semi-formalmente el comportamiento interno de cada subsistema, de forma que se pudieran añadir mecanismos de valiación y verificación incluso en tiempo de ejecución.
- Debía servir para crear sistemas que, dentro de una arquitectura orientada a servicios, se basaran en la composición de servicios.
- Se debía desarrollar una infraestructura de soporte y un entorno de desarrollo para facilitar su adopción de forma práctica en la industria.

2 Cervantes Framework

Tomando como base estas necesidades, se desarrolló una infraestructura básica para facilitar el despliegue de sistemas distribuidos, concebidos como subsistemas que intercambian información mediante mensajes asíncronos y los procesan mediante workflows como máquinas de estados. Su arquitectura básica se compone de los siguientes bloques, tal y como se muestra en la Figura 1 a):

- *Core Framework*: contiene los elementos comunes a toda la infraestructura como mecanismos de activación y parada, registro de métricas, manejo común de excepciones y logger.
- *Control Framework*: encapsula la lógica de actuación ante la llegada de un mensaje. Se compone de un gestor denominado *Manager* que implementa las funciones de un consumidor, que duerme hasta recibir una petición y va encolando las mismas cuando se encuentra en ejecución. Cada mensaje es procesado mediante un workflow, que en la implementación actual está basado en máquinas de estados.
- *Boundary Framework*: responsable de establecer la comunicación con otros sistemas mediante mensajes asíncronos independientes del protocolo (ej. XML, XML-RPC, SOAP, Java .RMI, .NET Remoting, etc). Está compuesto de un *Sender* responsable del envío y de un *Server* que recibe la información en un formato dado y lo convierte a la estructura general de mensajes internos del sistema.
- *Entity Framework*: almacena la información de los diversos workflows en ejecución en un espacio de memoria compartida independiente para cada subsistema y mantiene un espejo de los mismos en medio persistente. El sistema de memoria hace la función de pizarra compartida, siendo accesible en modo lectura y escritura por las diversas máquinas de estados del sistema.

En base a la arquitectura anteriormente expuesta, el comportamiento dinámico de la infraestructura sigue la siguiente secuencia de pasos, tal y como se muestra en la Figura 1 b). 1) Se recibe una petición, que llega a través de un protocolo determinado y es atendida por el *Server* correspondiente. La información es traducida al sistema de mensajes asíncrono interno. 2) El mensaje interno transformado es enviado al *Manager*. 3) El mensaje es almacenado en la cola de

peticiones (que está ordenada por un conjunto de valores que representan prioridades). 4) Atendiendo a esa cola de mensajes ordenada por prioridad, el mensaje siguiendo el patrón consumidor es procesado. 5) Si se trata de una petición, es necesario crear una máquina de estados que ejecute la lógica de la solicitud, y pasaría a formar parte del anillo de workflows denominado *SMList*. En el caso de que se tratara de un mensaje respuesta, el sistema salvaguarda quién solicitó la petición, de forma que la respuesta llega automáticamente a a la máquina de estados que estaba en ejecución en un estado *esperando la respuesta*. 6) La máquina de estados forma parte de la lista de workflows en ejecución y espera la llegada del *Token Ring* para poder comenzar. 7) Cuando le toca el turno de ejecución, comienza la lógica del workflow, pudiendo acceder a la estructura de datos y enviar mensajes a otros subsistemas. 8) Generalmente toda petición lleva asociada una respuesta, por lo que como resultado final de la ejecución del workflow, el subsistema invocado remitirá una respuesta final. Para enviar los resultados se utiliza el mismo sistema de mensajería, por lo que el mensaje resultado será trasladado al *Manager*, que en este caso hará de simple *proxy* hacia el módulo *Boundary*. 9) Llegado el mensaje al *Boundary Framework*, quedará almacenado en el *Sender* correspondiente y cuando se hayan enviado los anteriores el mensaje será emitido. Tras terminar esta operación la máquina de estados que solicitó el envío es informada para que finalice del todo su ejecución.

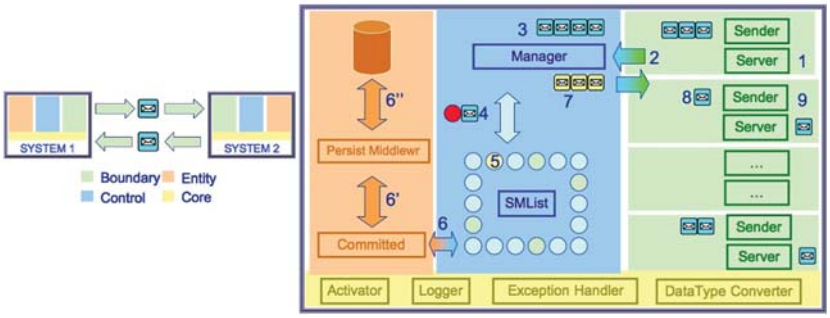


Fig. 1. a) Componentes del CERVANTES Framework; b) Pasos de ejecución

Posteriormente se desarrolló una herramienta para diseñar sistemas que fueran a ser desplegados en la infraestructura de Cervantes. Se trata de un plugin basado en el Eclipse Modeling Framework(EMF)[4] y en el Graphical Modeling Framework (GEF)[5], que permite a los ingenieros del software modelar los sistemas, creando las máquinas de estados y los mensajes y generando de forma automática el código para diversas plataformas como Java y C#.NET.

3 Conclusiones

En este trabajo se ha presentado CERVANTES, un marco para el diseño y desarrollo de sistemas distribuidos. Se compone de una infraestructura de ejecución y un entorno de desarrollo integrado que, siguiendo un enfoque basado por modelos, permite la generación automática de código en diversas plataformas. Este

marco se creó en el año 2001, cuando todavía los conceptos relacionados con *MDE* y *SOC* no estaban ampliamente desarrollados en la literatura y ha tenido desde su inicio una clara componente de aplicación práctica en la industria.

Como demostración se plantea partir de un sistema de gestión de flotas distribuido y añadir alguna nueva funcionalidad, haciendo uso del Studio para diseñar los nuevos mensajes y máquinas de estados y desplegarlos en la infraestructura de ejecución. También se propone recoger las métricas y analizarlas dentro del Studio como mecanismo para identificar problemas en el despliegue de los sistemas distribuidos.

Como trabajo futuro planteamos los siguientes pasos: 1) Integrar un soporte metodológico que cubra todo el ciclo de vida del desarrollo, permitiendo guardar una trazabilidad desde los requisitos hasta la ejecución. Para ello planteamos extender la metodología NDT[6] para el desarrollo de sistemas distribuidos. 2) Refactorizar el Studio para seguir un enfoque basado en modelos, valorando la opción de utilizar SOAML[7]. 3) Incorporar la generación automática de casos de prueba en etapas tempranas[8]. 4) Integrar aspectos de interacción persona ordenador en el diseño de sistemas distribuidos, valorando la opción de utilizar IFML[9]. 5) En base a las métricas de ejecución recogidas, dotar de inteligencia a la infraestructura para conseguir la reconfiguración automática de los sistemas.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto NDTQ Framework (TIC-5789) de la Junta de Andalucía y el proyecto MEGUS del Ministerio de Economía y Competividad (TIN2013-46928-C3-3-R).

References

1. Schmidt, D.C.: Model-driven engineering. (2006)
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: Service-oriented computing: a research roadmap. International Journal of Cooperative Information Systems **17**(02) (2008) 223–255
3. Tsai, W.T.: Service-oriented system engineering: a new paradigm. In: Service-Oriented System Engineering, 2005. SOSE 2005. IEEE International Workshop, IEEE (2005) 3–6
4. EMF: Eclipse modeling framework project. Eclipse Foundation (2014) http://www.eclipse.org/modeling/emf/?project=emf, last accessed 04/2014.
5. GMF: Graphical modeling project. Eclipse Foundation (2014) http://eclipse.org/gmf-tooling/, last accessed 04/2014.
6. Escalona, M.J., Mejías, M., Torres, J., Reina, A.: The ndt development process. In: Web Engineering. Springer (2003) 463–467
7. OMG: Omg's service oriented architecture modeling language (soaml), http://www.omg.org/spec/soaml, accessed 2014-04-11
8. Gutiérrez, J.J.: Generación de pruebas del sistema a partir de la especificación funcional. PhD thesis, Universidad de Sevilla (2011)
9. OMG: Omg's interaction flow modeling language (ifml), http://www.ifml.org, accessed 2014-04-10

Análisis de la aplicabilidad de medidas software para el diseño semi-automático de arquitecturas

Aurora Ramírez, José Raúl Romero y Sebastián Ventura

Dpto. de Informática y Análisis Numérico
Universidad de Córdoba, Campus de Rabanales, 14071 Córdoba
{aramirez, jrromero, sventura}@uco.es

Resumen. Dar soporte semi-automático al ingeniero software durante la fase de diseño arquitectónico es una tarea compleja, pues se deben manejar múltiples alternativas de diseño que satisfagan criterios de calidad de interés para el ingeniero. Cuando en el contexto de los sistemas de soporte a la decisión se emplean técnicas metaheurísticas, estos criterios deben ser trasladados a medidas precisas que sean computacionalmente evaluables sobre las soluciones generadas automáticamente. Este trabajo analiza la problemática asociada a la utilización de medidas software en el ámbito del diseño de arquitecturas basadas en componentes mediante técnicas de optimización y búsqueda. Para ello se han analizado medidas relacionadas con los dos atributos de calidad más relevantes para este tipo de arquitecturas: la adecuación funcional y la mantenibilidad. El estudio realizado muestra que, si bien existe una gran variedad de medidas software, aún quedan diferentes aspectos que pueden limitar su aplicabilidad en el diseño semi-automático de arquitecturas.

Palabras clave: Diseño arquitectónico, ingeniería del software basada en búsqueda, medidas software

1. Introducción

Durante el diseño arquitectónico del software, los ingenieros no sólo se enfrentan a la identificación y especificación de los principales bloques funcionales que deben constituir el sistema, sino que es importante considerar cómo estos, y sus interacciones, van a repercutir en la calidad del software resultante. Para ello necesitan conocer la influencia de sus decisiones arquitectónicas en los requisitos no funcionales (RNF) desde las fases iniciales del proyecto [1].

La construcción de sistemas de soporte a la decisión (*Decision Support Systems*, DSS) es una opción interesante en este contexto [2]. En concreto, la obtención de modelos computacionales que automaticen la búsqueda de múltiples alternativas arquitectónicas es una área de investigación emergente, donde destaca la utilización de técnicas metaheurísticas [3]. Con el objetivo de resolver problemas de optimización, estas técnicas ofrecen procedimientos de búsqueda "inteligente" que son guiados por funciones de evaluación (*fitness*). Estas funciones proporcionan un valor numérico representativo de la calidad de las soluciones que el algoritmo trata, generalmente, de maximizar. Así, el algoritmo es capaz de comparar diferentes soluciones durante un proceso iterativo en el que se crean nuevas soluciones a partir de la información de las anteriores [4].

La aplicación de metaheurísticas a tareas de ingeniería del software (IS) se enmarca dentro del área denominada ingeniería del software basada en búsqueda (*Search Based Software Engineering*, SBSE) [5]. En este contexto, la importancia de las medidas cuantificables se hace patente, pues son necesarias para formular la función de *fitness* que guiará la búsqueda [6]. No obstante, es habitual que el propio diseñador del algoritmo sea quien defina dicha función, por lo que suele estar muy influenciada por la forma en la que la tarea de IS es reformulada como un problema de optimización. Como el arquitecto no interviene directamente en el proceso de búsqueda, los métodos clásicos de evaluación de arquitecturas basados en la observación también pierden su eficacia [7]. Por todo ello, la utilización de medidas software precisas es la única herramienta disponible para abordar de manera efectiva la evaluación automática de estas soluciones.

Además, se debe garantizar que dichas medidas sean representativas de los RNF de interés para el ingeniero software y, preferiblemente, estén formalizadas según las recomendaciones de un estándar de calidad del software [8]. Atributos como la adecuación funcional (el grado en el cual el sistema satisface la funcionalidad requerida) y la mantenibilidad (la capacidad del sistema para ser modificado) son criterios considerados por los arquitectos software cuando abordan la especificación y rediseño de arquitecturas basadas en componentes. Si bien estas tareas están siendo abordadas en SBSE, no existe consenso ni una formalización adecuada en lo referente a las medidas empleadas y su relación con dichos atributos.

Este trabajo aborda el análisis de la aplicabilidad de estos atributos de calidad y medidas software a la hora de afrontar tareas de optimización arquitectónica, bajo el paradigma de SBSE, para el desarrollo de un DSS. En concreto, se han recopilado y analizado 62 medidas software para la evaluación cuantitativa de arquitecturas basadas en componentes [9], estudiando aspectos como su formulación o los elementos arquitectónicos que evalúan. Analizando estos criterios es posible identificar las principales dificultades que presentan estas medidas a la hora de aunar la visión del arquitecto que utiliza el DSS y del algoritmo de búsqueda que internamente evalúa las soluciones.

El resto del artículo se estructura como sigue. La Sección 2 centra la problemática desde la perspectiva de las distintas áreas de interés y discute qué criterios han guiado este estudio. La Sección 3 recoge el catálogo de medidas seleccionadas, mientras que el análisis de su aplicabilidad se realiza en la Sección 4. Finalmente, la Sección 5 presenta las conclusiones y el trabajo futuro.

2. Problemática y criterios de análisis

La construcción de DSS para el diseño arquitectónico debe tener muy presente los intereses del ingeniero software. Por tanto, es necesario considerar la información que éste utiliza habitualmente para tomar sus decisiones, así como los objetivos que se persiguen, la tarea que se aborda o la madurez del diseño [10]. Si el DSS trabaja con metaheurísticas, considerar estos factores es aún más determinante pues, aunque los arquitectos no estén familiarizados con estas técnicas, son ellos

los que realmente determinarán los objetivos a optimizar. Hemos detectado tres áreas diferentes que confluyen en este ámbito, tal y como se resume en la Fig. 1.

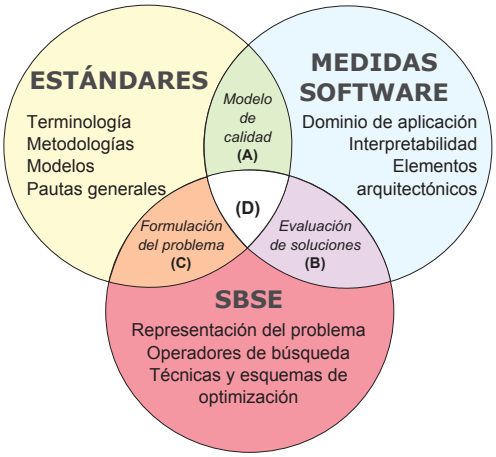


Fig. 1: Áreas de interés en el diseño semi-automático de arquitecturas

En primer lugar, para obtener medidas software que realmente sean de utilidad para los arquitectos es conveniente atenerse a un modelo de calidad [8], que actúe como marco de referencia y las dote de un significado preciso (zona A). Si bien estos modelos tienden a ser generales, pueden y deben especializarse para obtener un conjunto de medidas software bien definidas a partir de los factores de calidad que se quieren evaluar [11] o de los elementos arquitectónicos sobre los que se aplicarán [12,13]. Cabe destacar que aunque estos estándares ofrecen un proceso metodológico de evaluación, mantienen una visión clásica del proceso, donde es el ingeniero quien evalúa la arquitectura software ya desarrollada. Además, los estándares para la descripción de arquitecturas (ISO/IEC 42010 [14]) y su proceso de evaluación (ISO/IEC 42030), este último aún en desarrollo, quedan lejos de ofrecer medidas concretas cuantificables y específicas para evaluar su calidad.

Por otro lado, la utilización de medidas software es un aspecto de gran interés en la definición de las funciones de evaluación de las metaheurísticas (zona B). Actualmente, las funciones de *fitness* son mayoritariamente creadas de forma específica para los modelos de optimización propuestos, influenciados por el ámbito de aplicación [15,16] y la representación computacional interna de los elementos arquitectónicos que manejan [17]. Al estar determinadas por aspectos ajenos a los criterios del arquitecto, será difícil que éste llegue a comprender su significado, así como su repercusión en la calidad de las soluciones arquitectónicas resultantes. Así pues, es necesario abordar la recopilación y análisis de medidas ya existentes en el ámbito de la IS que, si bien pueden requerir alguna adaptación, permitan la definición de funciones de *fitness* más apropiadas.

Centrándose en la zona C, parece evidente que la utilización de modelos y terminología precisa, extraída de los estándares de la IS [8], supone un beneficio para la comunidad de SBSE, ya que va a permitir acercar sus resultados a los ingenieros software, logrando así una mayor interacción entre todos los implicados. Una de las mayores ventajas de esta unión es que permite a los expertos en metaheurísticas formular adecuadamente el problema de optimización [3].

A partir de los puntos de unión entre las áreas de interés, detectamos una zona común a todas ellas (D), que representa la problemática objeto de este estudio, y que hemos caracterizado bajo los siguientes aspectos:

1. *Representación de los elementos arquitectónicos.* Los elementos arquitectónicos que debe manejar el DSS, y cómo estos son especificados, van a influir en la selección de las medidas software que se pueden aplicar en el proceso de búsqueda. En este sentido, lo ideal es que se consideren notaciones y modelos ampliamente aceptados en la IS. Para la problemática que nos ocupa, se pretende discutir qué elementos arquitectónicos concretos son objeto de evaluación, y con qué nivel de granularidad, así como comprobar si requieren algún tipo de representación específica.
2. *Ámbito de aplicación.* Es posible asociar el objetivo del problema de optimización a los requisitos no funcionales que guian el proceso de decisión del ingeniero software. Por ejemplo, es habitual relacionar aspectos como el coste o la fiabilidad al diseño de arquitecturas orientadas a servicios, mientras que un factor como la reutilización es el principal objetivo en el diseño orientado a componentes. Este estudio se centra en la creación, rediseño y descubrimiento de arquitecturas basadas en componentes, pues son tareas de interés en la etapa de análisis, donde el arquitecto software necesitará un mayor soporte a la decisión.
3. *Aplicabilidad de medidas software existentes.* Debe evaluarse si las medidas software actualmente propuestas en la comunidad de IS son directamente transferibles al ámbito del diseño semi-automático. Obsérvese que muchas de estas medidas se basan en la observación por parte del arquitecto software. Sin embargo, para este caso de estudio será necesario seleccionar aquellas medidas cuantitativas que puedan ser calculadas computacionalmente sobre los propios modelos arquitectónicos.
4. *Consideración de los factores de calidad evaluables.* Puesto que el ingeniero se encuentra en las primeras etapas de análisis, no dispone de información precisa sobre cómo sus decisiones van a influir en el sistema software resultante. Es necesario considerar qué aspectos de la calidad del software son evaluables a este nivel de abstracción, y cómo se pueden reflejar en medidas cuantificables concretas.

   Considerando que el análisis planteado se centra en las fases tempranas del diseño arquitectónico basado en componentes, se van a abordar los siguientes criterios definidos en el estándar ISO/IEC 25010 [8]: la adecuación funcional (*functional suitability*), cuya subcaracterística del mismo nombre permite evaluar en qué grado los diseños generados y las alternativas arquitectónicas cumplen con la funcionalidad exigida; y la mantenibilidad (*maintainability*),

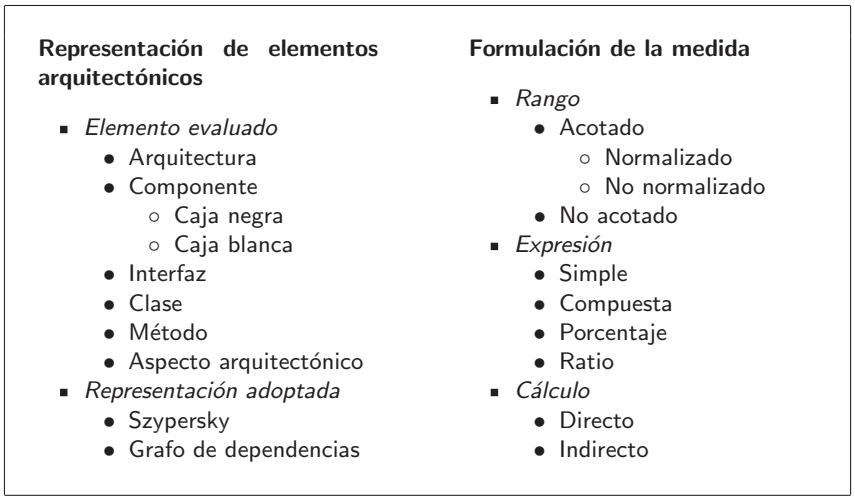


Fig. 2: Clasificación propuesta para el análisis de aplicabilidad de medidas

donde dos subcaracterísticas, como la modularidad (*modularity*) y la reutilización (*reusability*), son conceptos altamente relacionados con el diseño orientado a componentes.

5. *Definición precisa de las medidas software.* Aunque no es un requisito esencial a la hora de aplicar técnicas metaheurísticas, es conveniente que las medidas se definan en un intervalo bien acotado, preferiblemente normalizadas. Esto permitirá una mejor comprensión de su significado y facilitará el tratamiento computacional de los resultados en un tiempo aceptable.

   Para estudiar las implicaciones que tiene la formulación de las medidas en su aplicabilidad e interpretabilidad, en este estudio se considerarán como *expresiones simples* aquellas medidas basadas en contabilizar la ocurrencia de algún elemento arquitectónico, mientras que se denominará *compuestas* a aquellas que requieren la agregación de varios valores. También se pretende diferenciar entre las medidas que pueden obtenerse de un modo *directo* o *indirecto*, donde éstas últimas requieren información adicional al modelo arquitectónico o algún tipo de configuración por parte del arquitecto.

Ante la cantidad de medidas recopiladas y la variedad existente en cuanto a su formulación y a los elementos arquitectónicos evaluados, se hace necesario establecer una clasificación que permita abordar su estudio de forma conjunta y ordenada. La Fig. 2 muestra la clasificación propuesta, donde se especifican los diferentes aspectos observados en el conjunto de medidas que serán utilizados para abordar el análisis de su aplicabilidad.

3. Catálogo de medidas seleccionadas

La Tabla 1 recopila aquellas medidas que permiten evaluar la adecuación funcional de una arquitectura software y que cumplen con los criterios establecidos en la Sección 2. Se observa que para establecer el grado de satisfacción de la funcionalidad en el sistema es necesario que el método de evaluación disponga de la información referida a la funcionalidad esperada y la que realmente es cubierta en una fase de desarrollo concreta. Por ello, las medidas aquí mostradas asocian los elementos arquitectónicos presentes en la arquitectura con información adicional referida a los requisitos del sistema. En concreto, las medidas propuestas en [18] especifican la funcionalidad esperada mediante aspectos arquitectónicos (*concerns*), los cuales permiten identificar y separar las partes de interés del problema de diseño, mientras que las definidas en [19] se fundamentan en las operaciones e interfaces requeridas en el sistema.

Tabla 1: Medidas software para la adecuación funcional

Coverage [19]	D: Funcionalidad requerida que es cubierta por el componente E_a: Componente (N), interfaz F: $100 * \frac{Needed\,Interfaces \cap Provided\,Interfaces}{Needed\,Interfaces} \in [0, 100]$
Service Implementation Coverage [19]	D: Operaciones implementadas respecto a las especificadas E_a: Componente (N), operación F: $100 * \frac{Implemented\,Operations}{Specified\,Operations} \in [0, 100]$
$CDAC$ [18]	D: N^o de componentes asociados a un aspecto arquitectónico E_a: Aspecto arquitectónico, componente F: $\#Components \in [0, \#componentes_{arq}]$
$CDAI$ [18]	D: N^o de interfaces asociadas a un aspecto arquitectónico E_a: Aspecto arquitectónico, interfaz F: $\#Interfaces \in [0, \#interfaces_{arq}]$
$CDAO$ [18]	D: N^o de operaciones asociadas a un aspecto arquitectónico E_a: Aspecto arquitectónico, operación F: $\#Operations \in [0, \#operaciones_{arq}]$

D: Descripción E_a: Elemento arquitectónico F: Formulación
(N): Componente "de caja negra" (B): Componente "de caja blanca" (G): Grafo de dependencias

Las Tablas 2 y 3 recogen las medidas que afectan, en mayor medida, a la modularidad del sistema. Habitualmente, esta propiedad del software está relacionada con el grado de independencia entre los componentes que conforman la arquitectura, de forma que muchas de estas medidas se basan en el análisis de sus dependencias e interacciones. También es habitual considerar el tamaño de los componentes, ya que se debe evitar la presencia de componentes críticos que acaparen gran parte de la funcionalidad.

Finalmente, las medidas mostradas en las Tablas 4 y 5 están relacionadas con la reutilización. A diferencia del caso anterior, existe una mayor diversidad

en cuanto a los elementos arquitectónicos sobre los que se evalúa este criterio. La mayoría de las medidas se centran en definir la complejidad de los componentes o sus interfaces, pues la simplicidad a la hora de definir la funcionalidad ofrecida por esos componentes va a favorecer que puedan ser utilizados en otros sistemas. Otras medidas seleccionadas se centran también en evaluar la reutilización sobre diferentes dominios, por lo que será necesario hacer explícita esta información para que pueda ser manejada automáticamente.

Tabla 2: Medidas software para la modularidad (I)

TNC [20]	D: Nº de componentes en el sistema E_a: Arq., comp. F: $\#components$
CDT [20]	D: Nº de niveles de descomposición de la arquitectura E_a: Arq., comp. F: $\#levels$
WCT [20]	D: Máxima anchura de descomposición de la arquitectura E_a: Arq., comp. F: $max(\#components_{level})$
$CRIT_{link}$ [21]	D: Nº de componentes que exceden un nº máximo de enlaces E_a: Arq. (G), comp. F: $\#link\ component \in [0, \#comp._{arq}]$
$CRIT_{bridge}$ [21]	D: Nº de componentes "puente" (enlazan otros componentes) E_a: Arq. (G), comp. F: $\#bridge\ component \in [0, \#comp._{arq}]$
$CRIT_{inher}$ [21]	D: Nº de componentes "raíz" E_a: Arq. (G), comp. F: $\#root\ component \in [0, \#comp._{arq}]$
$CRIT_{size}$ [21]	D: Nº de componentes que exceden un tamaño crítico E_a: Arq. (G), comp. F: $\#component\ size \in [0, \#comp._{arq}]$
CC [22]	D: Complejidad basada en interfaces (IF_i) y tipos de datos (CM_j) E_a: Arquitectura F: $\sum_i^n IF_i * \sum_j^m CM_j$
$CI_{\%}$ [23]	D: Ratio entre interacciones de entrada (I_i) y salida (I_o) E_a: Arquitectura F: $\frac{I_i}{I_o} * 100 \in [0, 100]$
$I\%MCI$ [23]	D: Densidad de interacción en el sistema (ver $CI_{\%}$) E_a: Arquitectura F: $\frac{I_i+I_o}{CI_{\%}}$
CI [23]	D: Complejidad de las interacciones en el sistema E_a: Arquitectura F: $\frac{C_i+C_o}{C}$, $C = \#components_{arq}$
CDM [24]	D: Nº de interacciones (*paths*) entre componentes E_a: Arquitectura (G), componente F: $\sum_i^n \sum_j^n Path(i,j) \in [0, n(n-1) \sum_{i=0}^{n-2} \frac{(n-2)!}{(n-2-i)!}]$
$DCOM$ [24]	D: Ratio entre el número de interaciones actual y máximo E_a: Arq. (G), comp. F: $\frac{CDM}{CDM_{max}} \in [0, 1]$
$CIDM$ [24]	D: Nº medio de interacciones (I) por componente (N) E_a: Arq. (G), comp. F: $\frac{I}{N}$
TNL [20]	D: Nº de "links" en el sistema E_a: Arq., conect. F: $\#links$

D: Descripción E_a: Elemento arquitectónico F: Formulación
(N): Componente "de caja negra" (B): Componente "de caja blanca" (G): Grafo de dependencias

Tabla 3: Medidas software para la modularidad (II)

TNI [20]	D: N° de interfaces en el sistema E_a: Arquitectura, interfaz F: $\#interfaces$
$ANMC$ [20]	D: N° medio de métodos por componente E_a: Arq., comp., mét. F: $\frac{\#methods}{\#components}$
$ANLC$ [20]	D: N° medio de "links" entre componentes E_a: Arq., comp., con. F: $\frac{\#links}{\#components}$
$ANIC$ [20]	D: N° medio de interfaces por componente E_a: Arq., comp. (N), int. F: $\frac{\#interfaces}{\#components}$
$ANLI$ [20]	D: N° medio de "links" por interfaz E_a: Arq., con., int. F: $\frac{\#links}{\#interfaces}$
Afferent Coupling [18]	D: N° de componentes de los cuales requiere servicios un componente E_a: Componente (N) F:$\#components\ afferent\ relation \quad \in [0, n], \quad n = \#componentes_{arq}$
Efferent Coupling [18]	D: N° de componentes a los cuales provee servicios un componente E_a: Componente (N) F: $\#components\ efferent\ relation \quad \in [0, n], \quad n = \#componentes_{arq}$
LCC [18]	D: N° de aspectos (c) distintos asociados a un componente (i) E_a: Componente (N), aspecto arquitectónico F: $\#c_i + \#c_{inter_i} + \#c_{oper_i}$
Afferent Coupling [25]	D: N° de clases de las cuales dependen las clases del componente E_a: Comp. (B), clase F:$\#afferent\ rel_{A,B} \in [0, \#clases_{arq}]$
Efferent Coupling [25]	D: N° de clases externas que dependen de las clases del componente E_a: Comp. (B), clase F: $\#efferent\ rel_{A,B} \in [0, \#clases_{arq}]$
Coupling [25]	D: Porcentaje de relaciones entre clases de dos componentes E_a: Comp. (B), clase F: $\frac{R(A,B)}{R(A,all)} \in [0, 1]$
Instability [25]	D: Ratio entre el acoplamiento aferente y eferente E_a: Comp. (B), clase F: $\frac{C_e(A,B)}{C_e(A,B)+C_a(A,B)} \in [0, 1]$
Interface Violations [25]	D: Porcentaje del n° de accesos entre clases sin utilizar interfaces E_a: Comp. (B), clase, inter. F: $\frac{RI(A,B)}{R(A,all)} \in [0, 1]$
$CIBC$ [18]	D: N° de aspectos asociados a algún componente en común E_a: Aspecto arquitectónico, componente F: $\#concerns\ component\ interlacing, \in [0, \#apectos_{arq}]$
$IIBC$ [18]	D: N° de aspectos asociados a alguna interfaz en común E_a: Aspecto arquitectónico, interfaz F: $\#concerns\ interface\ interlacing, \in [0, \#aspectos_{arq}]$
$OOBC$ [18]	D: N° de aspectos asociados a alguna operación en común E_a: Aspecto arquitectónico, operación F: $\#concerns\ operation\ overlapping, \in [0, \#aspectos_{arq}]$

D: Descripción E_a: Elemento arquitectónico F: Formulación
(N): Componente "de caja negra" (B): Componente "de caja blanca" (G): Grafo de dependencias

Tabla 4: Medidas software para la reutilización (I)

CPD [21]	D: Densidad de elementos que constituyen la arquitectura E_a: Arq., comp. (B) F: $\frac{\#<constituent>}{\#components} \in [1, \#<constituent>]$
$IFCI$ [26]	D: Complejidad media de las interfaces ($IFC = V(GI_j)$ [22]) E_a: Arq., int. F: $\frac{IFC}{n}$
$ITCI$ [26]	D: Complejidad media de las interacciones (ITC) en función de la frecuencia (IF_{ij}) y tipos de datos intercambiados (CM_{ijk}) E_a: Arquitectura F: $\frac{ITC}{n}, ITC = \sum_{i=1}^{l}\sum_{j=1}^{m} IF_{ij} * \sum_{k=1}^{n} CM_{ijk}$
CP [26]	D: Complejidad del sistema en función del nº componentes ($C_{i,j}$) de cada tipo de complejidad ($W_{i,j}$) y un ajuste (VAF) E_a: Arquitectura F: $VAF * \sum_{i=1}^{3}\sum_{j=1}^{3} C_{i,j} * W_{i,j}$
CPC [27]	D: Complejidad en función del número de clases e interfaces ($CmpC$) y la complejidad de clases (CC) y métodos (MC) E_a: Componente (B) F: $CmpC + \sum_{i=1}^{n} CC_i + \sum_{j=1}^{m} MC_i$
CSC [27]	D: Complejidad basada en las clases que conforman el componente y las relaciones entre ellas E_a: Componente F: $\sum_{i=1}^{m}(Count(R_i) * W(R_i))$
CDC [27]	D: Complejidad basada en el nº de mensajes entre clases E_a: Componente F: $\sum_{i=1}^{n} DC(IM_i)$
CR [27]	D: Ratio entre el nº de métodos reutilizables en varios dominios (CCM_i) y el total de métodos especificados (CIM_j) E_a: Componente F: $\frac{\sum_{i=1}^{n} Count(CCM_i)}{\sum_{j=1}^{m} Count(CIM_j)}, \in [0,1]$
CRL_{Func} [27]	D: Ratio entre la funcionalidad soportada y la requerida E_a: Componente F: $\frac{\#functions\ supported}{\#functions\ required}$
CCM [28]	D: Complejidad basada en las relaciones entre clases (RCM), atributos (ACM), métodos (MCM) y mensajes ($MPCM$) E_a: Componente (B) F: $\alpha RCM + \beta ACM + \gamma MCM + \eta MPCM$
IC_i [22]	D: Complejidad basada en los tipos de interfaces (ILF, ELF) E_a: Componente (B) F:$\sum_{i=1}^{n} ILF_i + \sum_{j=1}^{m} EIF_j$
$V(GT_i)$ [22]	D: Complejidad basada en las interfaces ($V(GI)$) E_a: Componente (B) F: $\sum_{j=1}^{n} V(GI_j)$
$A(GT_i)$ [22]	D: Complejidad basada en restricciones ($V(GT_i)$) y número de operaciones (NO_i) E_a: Componente (B) F: $\frac{V(GT_i)}{NO_i}$
Complexity Ratio [19]	D: Nº medio de operaciones por interfaz proveída E_a: Comp. (N) F: $\frac{\#operations}{\#provided\ interfaces}$
Abstractness [25]	D: Ratio de elementos (S) abstractos en el componente E_a: Comp. (B) F: $\frac{card(abstract(S))}{card(S)} \in [0,1]$

D: Descripción E_a: Elemento arquitectónico F: Formulación

(N): Componente "de caja negra" (B): Componente "de caja blanca" (G): Grafo de dependencias

Tabla 5: Medidas software para la reutilización (II)

Ratio of method per Interface [29]	D: Nº medio de métodos por interfaz E_a: Comp. (N) F: $\frac{\#methods}{\#interfaces} \in [1, \#methods]$
NI [18]	D: Nº de interfaces E_a: Componente F: $\#interfaces$
NO [18]	D: Nº de operaciones E_a: Interfaz F: $\#operations$
APP [30]	D: Ratio entre el nº de argumentos (n_a) y métodos (n_p) E_a: Interfaz F: $\frac{n_a}{n_p}$
DAR [30]	D: Ratio entre el nº de pares parámetro-tipo (DAC) y el nº de argumentos (n_a) E_a: Interfaz F: $\frac{DAC}{n_a}, \in [0, 1]$
ARS [30]	D: Ratio entre el nº de apariencias de un par parámetro-tipo de dato (a) y el nº de argumentos E_a: Interfaz F: $\frac{\sum_{a \in A} \lvert a \rvert^2}{n_a}, \in [1, n_a]$
$V(GI_j)$ [22]	D: Complejidad basada en operaciones ($V(GO_k)$) E_a: Interfaz F: $\sum_{k=1}^{max} V(GO_k)$
Return values for method [29]	D: Nº medio de valores de retorno por método E_a: Método F: $\frac{\#return\ values}{\#methods} \in [0, 1]$
Methods without arguments [29]	D: Nº medio de métodos sin argumentos E_a: Método F: $\frac{\#methods\ without\ arguments}{\#methods} \in [0, 1]$
Method arguments for method [29]	D: Nº medio de argumentos por método E_a: Método F: $\frac{\#arguments}{\#methods}$
Constructors per class [29]	D: Nº medio de constructores por clase E_a: Clase F: $\frac{\#constructors}{\#classes}$

D: Descripción E_a: Elemento arquitectónico F: Formulación
(N): Componente "de caja negra" (B): Componente "de caja blanca" (G): Grafo de dependencias

4. Análisis de aplicabilidad

La Tabla 6 detalla el porcentaje de medidas recopiladas en el catálogo que cumplen con las distintas categorías establecidas en la Fig. 2. Las medidas descritas en trabajos en los que no se indica explícitamente la representación adoptada han sido asociadas a la categoría «no especificado». Estos resultados permiten valorar globalmente el conjunto de medidas disponibles respecto a los criterios de interés establecidos en la Sección 2.

Como puede observarse, una amplia mayoría de las medidas se centran en la evaluación de la arquitectura completa o de un componente. En el primer caso, las medidas van a ofrecer al DSS una valoración global de las soluciones. Por el contario, la evaluación individual de los componentes implica valorar parcialmente la solucion, de forma que será necesario considerar simultáneamente varias medidas en el proceso de optimización. Además, el arquitecto puede trabajar a

distintos niveles de granularidad gracias a la existencia de medidas específicas para otros elementos arquitectónicos, como los métodos y las clases.

Tabla 6: Resultados del análisis de aplicabilidad para la clasificación propuesta

Representación de elementos arquitectónicos		**Formulación de la medida**	
Elemento evaluado		*Rango*	
Arquitectura	38,71 %	Acotado	48,39 %
Componente	37,10 %	Normalizado	40,00 %
Caja negra	30,43 %	No normalizado	60,00 %
Caja blanca	65,22 %	No acotado	51,61 %
Interfaz	8,06 %	*Expresión*	
Clase	1,61 %	Simple	32,26 %
Método	4,84 %	Compuesta	19,35 %
Aspecto arquitectónico	9,68 %	Porcentaje	4,84 %
Definición adoptada		Ratio	43,55 %
Szypersky	54,84 %	*Cálculo*	
Grafo de dependencias	14,52 %	Directo	75,81 %
No especificado	30,64 %	Indirecto	24,19 %

Cuando el objetivo del DSS consiste en optimizar aspectos funcionales de una arquitectura, como es el caso que nos ocupa, las alternativas de diseño generadas se van a diferenciar entre sí por la forma en la que se distribuye la funcionalidad entre los componentes y la estructura global resultante. Por tanto, es necesario conocer la estructura interna del componente. En este sentido, cabe destacar que la mayoría de medidas consideran al componente como un elemento de «caja blanca», lo cual beneficia su aplicabilidad en este contexto.

Por otro lado, no existe homogeneidad en la forma en la que un componente es especificado pues, si bien la mayoría de medidas adoptan las definiciones de Szypersky, hay un amplio porcentaje de medidas para las que no se especifica claramente qué es un componente. También son frecuentes las propuestas basadas en el análisis de dependencias e interacciones, las cuales no son formalmente definidas. Otro ejemplo encontrado habitualmente es la evaluación basada en un grafo de dependencias, donde la representación adoptada implica una visión simplificada de la arquitectura en la que se pierde la identidad de las interfaces y los conectores, tratados como simples enlaces entre los componentes. En general, no disponer de medidas asociadas a una representación ampliamente aceptada va a dificultar su inclusión en el modelo de decisión, ya que van a establecer una fuerte dependencia con la forma en la que se representan internamente las soluciones. Esto perjudica a la obtención de un conjunto de medidas generales y aplicables entre las que pueda elegir el ingeniero software.

El desconocimiento de los intervalos entre los cuales varían las medidas es uno de los principales inconvenientes a la hora de interpretarlas. Sin una escala conocida y sin valores de referencia, las soluciones evaluadas por un algoritmo

de optimización carecerán de un significado preciso para el arquitecto software. Por otro lado, si se quiere dotar al DSS de la posibilidad de manejar simultáneamente alternativas arquitectónicas de distintos tamaños, estos límites deben ser constantes. Sin embargo, muchas de las medidas analizadas definen límites dependientes del número de componentes presentes en la arquitectura.

De forma similar, los valores obtenidos al utilizar medidas *compuestas* pueden no ser representativos para el ingeniero software, pues agregan en una misma expresión valores obtenidos sobre diferentes elementos arquitectónicos. En este sentido, la definición de medidas basadas en ratios, siempre que estén acotados, y porcentajes, serán de mayor utilidad.

Es importante resaltar que la mayoría de medidas analizadas son directamente aplicables sobre los elementos arquitectónicos que va a manejar el DSS. En el caso de que las medidas requieran información adicional, conviene que dicha información se incorpore de manera precisa en el modelo arquitectónico. Un caso más problemático es la definición de medidas con parámetros, ya que si no se ofrecen valores de referencia, el proceso de decisión estará influenciado por la opinión subjetiva del arquitecto software a la hora de configurarlos, aún cuando estos parámetros tienen una gran repercursión en el proceso de búsqueda.

Finalmente, no es frecuente que los autores propongan medidas directamente relacionadas con los criterios de calidad definidos en el estándar, sino que esta asociación debe realizarla el ingeniero. Más aún, no existe homogeneidad a la hora de definir los conceptos que subyacen a las medidas propuestas. Por citar algún ejemplo, dentro del conjunto de medidas analizadas es posible encontrar el término «acoplamiento» en diferentes contextos y a distinto nivel de granularidad. De forma similar, la complejidad de los componentes ha sido definida desde distintas perspectivas. Todo ello dificulta el proceso de selección de las medidas por parte del ingeniero software, ya que va a requerir un conocimiento profundo de cada medida propuesta y sus implicaciones a la hora de ser incluida en el proceso de optimización. Ambos hechos se alejan de lo que cabe esperar de un DSS, donde los intereses del arquitecto software en el proceso de optimización arquitectónica se establecen a nivel de requisitos no funcionales y bajo la utilización de los modelos y estándares con los que trabaja habitualmente.

5. Conclusiones y trabajo futuro

En este artículo se ha presentado la problemática que rodea a la definición y selección de medidas software en el contexto de la construcción de DSS en los que la búsqueda de alternativas de diseño se realiza mediante técnicas metaheurísticas. Encontrar una relación consistente entre los factores de calidad de interés para el ingeniero software y las medidas concretas que van a guiar la búsqueda automática de soluciones arquitectónicas constituye un aspecto esencial a la hora de acercar este tipo de modelos de decisión al arquitecto software.

A través de dos factores de calidad, la adecuación funcional y la mantenibilidad, se han analizado las distintas medidas software existentes y sus implicaciones a la hora de ser aplicadas a diferentes problemas de optimización

arquitectónica desde la perspectiva de SBSE. A su vez, se ha destacado la importancia de especificar adecuadamente los modelos arquitectónicos sobre los que se evalúan las medidas y los intervalos entre los cuales se definen, pues ambos factores facilitarán al arquitecto la configuración de los objetivos que va a perseguir el proceso de búsqueda, así como la interpretabilidad de los resultados obtenidos.

Este estudio constituye un primer acercamiento a la problemática planteada, el cual puede ser ampliado con el análisis de otros atributos de calidad, como la compatibilidad o la fiabilidad. Esto a su vez permitirá extender el ámbito de aplicación de los DSS a aspectos de integración y despliegue arquitectónico. Por otro lado, estudiar las dependencias entre los distintos atributos de calidad con el objetivo de crear un modelo de decisión basado en la optimización simultánea de varios criterios es también una línea de trabajo interesante.

Agradecimientos

Trabajo apoyado por el Ministerio de Ciencia y Tecnología, proyecto TIN2011-22408, y fondos FEDER.

Referencias

1. S. Mahmood, R. Lai, Y. S. Kim, J. H. Kim, S. C. Park, and H. S. Oh, "A survey of component based system quality assurance and assessment," *Information and Software Technology*, vol. 47, no. 10, pp. 693 – 707, 2005.
2. D. Falessi, G. Cantone, R. Kazman, and P. Kruchten, "Decision-making Techniques for Software Architecture Design: A Comparative Survey," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 33:1–28, 2011.
3. A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software Architecture Optimization Methods: A Systematic Literature Review," *IEEE Transactions on Software Engineering*, vol. 39, no. 5, pp. 658–683, 2013.
4. A. Ramírez, J. R. Romero, and S. Ventura, "Identificación de Componentes en Arquitecturas Software Mediante Programación Evolutiva," in *XVIII Jornadas en Ingeniería del Software y Bases de Datos (JISBD'13)*, pp. 413–426, 2013.
5. M. Harman, S. A. Mansouri, and Y. Zhang, "Search Based Software Engineering: Trends, Techniques and Applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 11:1–61, 2012.
6. M. Harman and J. A. Clark, "Metrics Are Fitness Functions Too," in *10th IEEE International Symposium on Software Metrics*, pp. 58–69, 2004.
7. L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 638–653, 2002.
8. ISO, *ISO/IEC 25010. Software engineering-Software product Quality Requirements and Evaluation (SQuaRE) Quality Model.* ISO, jul 2008.
9. M. Abdellatief, A. B. M. Sultan, A. A. A. Ghani, and M. A. Jabar, "A mapping study to investigate component-based software system metrics," *J. Syst. Soft.*, vol. 86, no. 3, pp. 587–603, 2013.
10. M. Nowak and C. Pautasso, "Goals, Questions and Metrics for Architectural Decision Models," in *6th International Workshop on SHAring and Reusing Architectural Knowledge (SHARK'11)*, pp. 21–28, ACM, 2011.

11. D. Perez-Palacin, R. Mirandola, and J. Merseguer, "On the relationships between QoS and software adaptability at the architectural level," *J. Syst. Soft.*, vol. 87, pp. 1 – 17, 2014.
12. F. Losavio, L. Chirinos, A. Matteo, N. Lévy, and A. Ramdane-Cherif, "ISO quality standards for measuring architectures," *J. Syst. Soft.*, vol. 72, no. 2, pp. 209 – 223, 2004.
13. A. Alvaro, E. Santana de Almeida, and S. Romero de Lemos Meira, "A Software Component Quality Framework," *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 1, pp. 1–18, 2010.
14. ISO, *ISO/IEC FDIS 42010/D9. Systems and software engineering - Architecture description.* ISO, mar 2011.
15. V. Cortellessa, F. Marinelli, and P. Potena, "An optimization framework for "build-or-buy"decisions in software architecture," *Computers & Operations Research*, vol. 35, no. 10, pp. 3090–3106, 2008.
16. P. Potena, "Optimization of adaptation plans for a service-oriented architecture with cost, reliability, availability and performance tradeoff," *J. Syst. Soft.*, vol. 86, no. 3, pp. 624–648, 2013.
17. O. Raiha, K. Koskimies, and E. Makinen, "Scenario-Based Genetic Synthesis of Software Architecture," in *Fourth International Conference on Software Engineering Advances (ICSEA '09)*, pp. 437–445, 2009.
18. C. Sant'Anna, E. Figueiredo, A. Garcia, and C. J. Lucena, "On the Modularity of Software Architectures: A Concern-Driven Measurement Framework," in *Software Architecture*, vol. 4758 of *LNCS*, pp. 207–224, Springer Berlin Heidelberg, 2007.
19. M. F. Bertoa and A. Vallecillo, "Quality attributes for COTS components," in *6th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, 2002.
20. N. Salman, "Complexity Metrics as Predictors of Maintainability and Integrability of Software Components," *Journal of Arts and Sciences*, vol. 5, pp. 39–50, 2006.
21. V. L. Narasimhan and B. Hendradjaya, "Some theoretical considerations for a suite of metrics for the integration of software components," *Information Sciences*, vol. 177, no. 3, pp. 844 – 864, 2007.
22. S. Mahmood and R. Lai, "A Complexity Measure for UML Component-based System Specification," *Softw. Pract. Exper.*, vol. 38, no. 2, pp. 117–134, 2008.
23. L. Kharb and R. Singh, "Complexity Metrics for Component-oriented Software Systems," *SIGSOFT Softw. Eng. Notes*, vol. 33, no. 2, pp. 4:1–3, 2008.
24. N. S. Gill and Balkishan, "Dependency and Interaction Oriented Complexity Metrics of Component-based Systems," *SIGSOFT Softw. Eng. Notes*, vol. 33, no. 2, pp. 3:1–3:5, 2008.
25. K. Krogmann, *Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis.* KIT Scientific Publishing, 2010.
26. T. Wijayasiriwardhane and R. Lai, "Component Point: A system-level size measure for Component-Based Software Systems," *J. Syst. Soft.*, vol. 83, no. 12, pp. 2456 – 2470, 2010.
27. E. S. Cho, M. S. Kim, and S. D. Kim, "Component metrics to measure component quality," in *8th Asia-Pacific Software Engineering Conference*, pp. 419–426, 2001.
28. F. Wu and T. Yi, "A Structural Complexity Metric for Software Components," in *1st Int. Symposium on Data, Privacy, and E-Commerce*, pp. 161–163, 2007.
29. M. F. Bertoa, J. M. Troya, and A. Vallecillo, "Measuring the Usability of Software Components," *J. Syst. Softw.*, vol. 79, pp. 427–439, Mar. 2006.
30. M. Boxall and S. Araban, "Interface metrics for reusability analysis of components," in *Australian Software Engineering Conference*, pp. 40–51, 2004.

Propuesta de metodología de despliegue de aplicaciones en nubes heterogéneas con TOSCA

Jose Carrasco, Javier Cubo, y Ernesto Pimentel

Universidad de Málaga
Departamento de Lenguajes y Ciencias de la Computación, España
{josec,cubo,ernesto}@lcc.uma.es

Resumen. Desplegar y controlar una aplicación compleja sobre un conjunto heterogéneo de proveedores es un problema muy novedoso y complejo al que los clientes de las plataformas de *cloud* se deben de enfrentar. Los proveedores exponen sus servicios de acuerdo a especificaciones independientes incurriendo en una falta de portabilidad e interoperabilidad que converge en la problemática conocida como *vendor lock-in*. Han surgido varias propuestas que aportan soluciones a este ámbito, como el estándar TOSCA que permite describir una aplicación y automatizar su despliegue de forma automática sobre un único proveedor. Extendiendo el estándar mencionado, en este trabajo proponemos una metodología de despliegue y orquestación de los componentes de una aplicación en un entorno *multi-cloud* mediante el uso simultáneo de servicios de diferentes proveedores.
Palabras clave: *Cloud Computing*; Despliegue multi-cloud; Componentes cloud; Orquestación de servicios; TOSCA.

1. Introducción

Cloud Computing [1] o Computación en la Nube ha pasado a ser un componente clave de la Internet, tal y como se conoce y explota actualmente. Este modelo es el resultado de la evolución y asociación de diferentes paradigmas y tecnologías (Virtualization, Client-Server-Model, Peer-to-Peer, Grid-Computing). *Cloud Computing* [2] es un término general para definir un modo de computación que implica un conjunto de componentes (normalmente servidores o bases de datos) conectados e interaccionando entre sí a través de la red, normalmente Internet, y hospedados en el *Cloud* o la Nube. Alojados en estas redes distribuidas se encuentran los diferentes sistemas que ofrecen su funcionalidad a través de servicios clasificados en diferentes niveles (*Infrastructure-as-a-Service*, IaaS; *Platform-as-a-Service*, PaaS; *Software-as-a-Service*, SaaS).

Los usuarios del *Cloud* pueden, por tanto, acceder a los recursos de computación de manera dinámica, flexible y escalable, transformando Internet en un mercado global de colaboración de servicios interconectados a través de la Nube. Por lo tanto, lo ideal sería que los usuarios pudieran desplegar sus desarrollos en cualquier entorno *cloud* ofertado por los diferentes proveedores existentes. No obstante, este es un escenario muy optimista que no se da en la realidad [3], ya que los proveedores han desarrollado sus plataformas sin seguir un estándar o

metodología común por lo que cada sistema necesita usar diferentes mecanismos para albergar desarrollos heterogéneos.

Desde el punto de vista de los desarrolladores esta falta de comunicación entre los proveedores se traduce en una falta de portabilidad de sus aplicaciones, lo cual puede tener consecuencias favorables o desastrosas, según el contexto. Cuándo se comienza el desarrollo de una aplicación se escoge un entorno *cloud* (precedido o no por un estudio más o menos exhaustivo) de manera que se integren las necesidades de la plataforma prácticamente desde el nacimiento de la aplicación, así como los requisitos de la aplicación en sí y los módulos o servicios que la constituyen. Además, esta relación puede ser bidireccional ya que los diferentes proveedores suelen ofertar dentro de sus plataformas diferentes servicios muy útiles para los desarrolladores, los cuales los integran y usan en sus sistemas realizando una unificación con el entorno que los convierte en dependientes del mismo, lo que se conoce como el problema del *vendor lock-in*. De esta manera el desarrollador contrae un bloqueo directo con la plataforma en la que se apoya el proyecto, quedando bloqueado ante futuros cambios por parte del proveedor ya que cuanto más evolucionada esté la aplicación más compleja será la tarea de portarla a otra plataforma.

De acuerdo a la descripción anterior parece complicado imaginar una aplicación compuesta por varios módulos distribuidos a lo largo de varias plataformas e interconectados entre ellos que corresponda al concepto de *multi-deployment*, o como vamos a denominarlo a partir de ahora, despliegue heterogéneo o multidespliegue [4, 5].

TOSCA [6] (*Topology and Orchestration Specification for Cloud Applications*) es un estándar de OASIS que propone ciertos mecanismos para permitir la portabilidad entre diferentes nubes, de forma más o menos transparente al usuario. La intención primordial es estandarizar el despliegue automático entre los diferentes servicios *cloud*, con la idea de hacer más fácil el uso de los mismos. Así, TOSCA permite una descripción interoperable muy expresiva de estos servicios: cómo están relacionados, su comportamiento operacional con independencia del proveedor que cree el servicio y la tecnología de alojamiento usada.

Aunque TOSCA afronta de forma sólida la problemática descrita [7], lo hace parcialmente mediante la distribución de los componentes de una aplicación a través de los servicios de un único *cloud*. En este trabajo presentamos una propuesta de metodología para llevar a cabo multidespliegues en plataformas *Cloud*. Para ello, nos basamos en el estándar mencionado anteriormente, TOSCA, con la intención de aportar nuevas características a su especificación en todos sus niveles, para permitir el despliegue interoperable y portable de las aplicaciones descritas en dicho estándar en entornos de *cloud* heterogéneos. De esta manera, proponemos ampliar la funcionalidad del estándar y su aplicación en distintos casos de uso. En nuestra propuesta, pretendemos definir nuevos mecanismos con el objetivo de gestionar la interacción de servicios o módulos de una aplicación *multi-cloud*, tomando como enfoque el concepto de orquestación de servicios basados en *Cloud*, y aplicando la metodología de composición y orquestación de servicios web [8].

El resto de este artículo está organizado según la siguiente estructura. La Sección 2 presenta una breve introducción del estándar TOSCA. La Sección 3 describe nuestra propuesta de multidespliegue, comparando en la Sección 4 otras alternativas disponibles. Finalmente, presentamos nuestras conclusiones en la Sección 5.

2. Breve introducción a TOSCA

La especificación de TOSCA determina un modelo para definir las aplicaciones o sistemas desarrollados y los servicios requeridos para su funcionamiento. Para ello, especifica una descripción de los elementos que componen una aplicación en los que se definen todas sus características, restricciones, naturaleza y comportamiento, (*Node Types*), tal y como se puede observar en la Fig. 1. Para expresar las relaciones entre los distintos elementos del sistema TOSCA establece un conjunto de relaciones (*Relationship Type*) que permiten determinar las conexiones y dependencias entre componentes (p. ej., conexión de una aplicación web y una base de datos). Estos elementos especifican las posibles entradas y salidas que pueden tener los módulos de la aplicación, a los que modelan, sin concretar sus valores finales. Además, para estructurar los *Types*, TOSCA define una jerarquía de tipos para la extensión y la clasificación de los mismos. En cuanto a su gestión, los *Types* definen operaciones que describen su comportamiento, por ejemplo, un servidor puede definir una operación de inicio para ponerlo en marcha y otra para permitir el despliegue de aplicaciones en su interior. Una vez modelados los elementos del sistema, TOSCA especifica la topología como una metodología para definir de forma exhaustiva la estructurar de una aplicación. Esta topología se compone de nodos y relaciones (ver Fig. 1) que no son más que los componentes concretos que forman la aplicación, los cuales están tipados de acuerdo a los tipos mencionados anteriormente. Todos estos elementos componen la Plantilla de Servicios (*Service Template*) de una aplicación descrita en TOSCA.

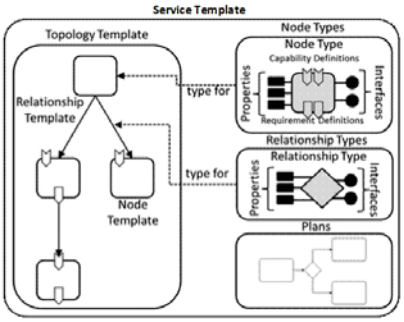


Fig. 1. Proceso de Metaorquestación. (Fuente de la imagen [6]).

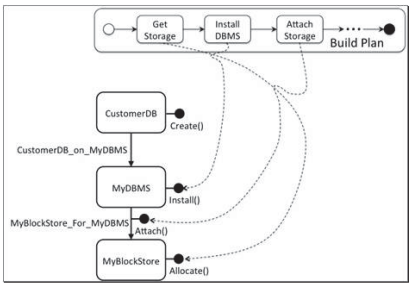


Fig. 2. Metodología de ejecución de planes definida por TOSCA. (Fuente de la imagen [9]).

Dentro de la plantilla de servicio encontramos también los planes de orquestación que especifican mediante un lenguaje de *workflow* (p.ej. BPEL, BPMN, etc.) el proceso de instanciación y gestión de los componentes expresados en la topología. Dichos planes especifican las llamadas de las operaciones de los componentes de la aplicación con el fin de instanciar la estructura descrita en la topología, como se puede observar en la Fig. 2. El modelado de una descripción basada en TOSCA se empaqueta en un archivo contenedor denominado CSAR (*Cloud Service Archive*). TOSCA especifica la estructura del CSAR para que cualquier plataforma que implemente el estándar pueda llevar a cabo la instanciación y orquestación de los componentes que describe, haciéndolo de esta forma portable entre los proveedores.

Para establecer un entorno homogéneo de plataformas *cloud* en el que desplegar los CSAR que modelan las aplicaciones, TOSCA define un conjunto de buenas prácticas en el que se describen los mecanismos necesarios que un proveedor debe de implementar para adaptar el estándar a los servicios ofertados en su plataforma [9], asegurando así la compatibilidad en el proceso de interpretación e instanciación de una aplicación modelada en un CSAR.

Aunque el estándar no cuenta con una implementación oficial, sí podemos encontrar una aproximación que soporta la mayoría de las características de TOSCA, OpenTOSCA [10]. Esta herramienta forma parte de un marco de trabajo que da soporte a todos los aspectos del estándar, definición y construcción de un CSAR [11], así como a la construcción de planes [12]. Sin embargo, esta aproximación plantea varios problemas de de cara la idea propuesta, como veremos en la Sección 4. Otros proveedores públicos (p.ej. Fujitsu, HP, IBM) ofertan sus adaptaciones del estándar [13], los cuales podrían ser utilizados para dar soporte a la idea que se presenta en este trabajo.

3. Propuesta de multidespliegue

En esta sección presentamos nuestra propuesta para realizar despliegues heterogéneos basado en TOSCA.

3.1. Ejemplo de motivación

Para ilustrar la idea presentada, mostramos en la Fig. 3 la distribución heterogénea de los componentes de la aplicación *Online Retailin* que simula un sistema de venta al por menos. En este escenario se despliegan los módulos web en la plataforma de un proveedor (HP) y las bases de datos en otro (Amazon). Además se muestra como se mantienen las conexiones entre módulos, ya estén desplegados en la misma plataforma o no, necesarias para asegurar el funcionamiento correcto del sistema.

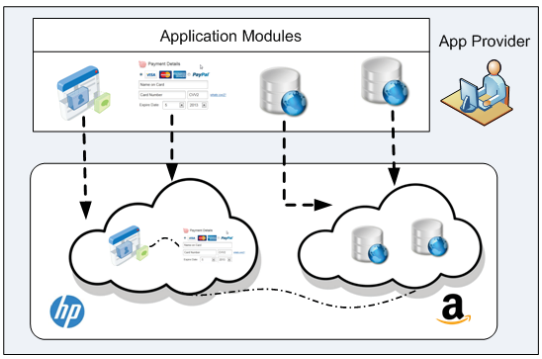


Fig. 3. Distribución heterogénea de la aplicación *Online Retailing*

Esta aplicación podría ser desplegada sobre cualquier proveedor en servicios IaaS o PaaS, por lo que tras la descripción del *vendor lock-in* de la Sección 1 podemos ver como es necesario tener en cuenta las cuestiones de portabilidad. Por otro lado, durante un despliegue heterogéneo hay que conservar la estructura topológica de la aplicación, manteniendo las relaciones y dependencias entre sus componentes, independientemente de los proveedores finales utilizados. Esto se ve afectado por la falta de interoprabilidad entre los proveedores, lo que complica la orquestación entre los servicios necesarios para desplegar los componentes de las aplicaciones y establecer las conexiones necesarias que tienen lugar en el despliegue sobre varios proveedores de forma simultanea.

Con el objetivo de afrontar la problemática de interoperabilidad, portabilidad y uso de contextos *cloud* heterogéneos, descrita en la Sección 1, proponemos una metodología basada en el estándar TOSCA para fomentar el desarrollo de aplicaciones que usen los servicios de múltiples plataformas de forma interoperable.

3.2. Proyección de una aplicación TOSCA

Dado un CSAR, una *proyección* sobre un proveedor *cloud* determina los elementos del mismo que van a ser distribuidos sobre los servicios de su plataforma (es una fragmentación del CSAR) en la que se agrupan todos sus componentes de acuerdo a los proveedores finales cuyos servicios son requeridos.Una proyección, por lo tanto, es una entidad contenedora atómica que alberga todos los componentes de la topología de una aplicación que se van a desplegar sobre un proveedor en concreto, y debe de describir de forma exhaustiva dichos componentes: sus restricciones, requisitos y capacidades, dependencias y relaciones, su gestión y comportamiento (orquestación), etc.

Al recibir un proveedor una proyección, está recibiendo los componentes que deben de ser desplegados en su plataforma junto a toda la información necesaria para llevar a cabo esta tarea. Así, distribuyendo cada una de las proyecciones entre sus plataforma correspondientes se lleva a cabo un proceso de despliegue en un entorno *cloud* heterogéneo donde cada plataforma recibe y contiene exclusivamente los elementos de la topología descritos en su proyección.

Al realizar las proyecciones es necesario mantener la estructura de la aplicación (topología) durante el despliegue de la misma. En este caso, los proveedores finales necesitan algún mecanismo, un plan, donde se les indique los pasos a seguir para realizar el despliegue y la gestión de los componentes que alberga una proyección. Así, la fragmentación de una plantilla de servicio también se realiza sobre su plan de orquestación (obteniendo un plan proyectado) para extraer la información que asegure el despliegue correcto de los componentes de una proyección, manteniendo sus relaciones y dependencias. Gracias a esto, un proveedor es capaz de instanciar los elementos de la topología que le corresponden siguiendo el plan de orquestación adaptado a ellos.

No debemos olvidar que componentes de una aplicación que están relacionados entre sí pueden ser proyectados sobre proveedores diferentes, por lo que es necesario establecer y mantener estas relaciones y dependencias entre plataformas

para asegurar la topología de la aplicación. Esto implica nuevas cuestiones de orquestación, recolección de datos y conexión de componentes que no contempladas en la definición original de TOSCA que pretendemos solventar en nuestra propuesta.

Por estos motivos, proponemos aprovechar la especificación del estándar de TOSCA y definir cada proyección como un CSAR, ya que es un contenedor lo suficiente expresivo como para satisfacer las cuestiones mencionadas. Por supuesto, al igual que sucede en el contexto de TOSCA, los proveedores deben de soportar el estándar (es decir, deben de ser TOSCA-*compliant*) mediante la adaptación de la especificación para poder recibir e interpretar a las distintas proyecciones de una aplicación, tanto los elementos de la topología como los planes proyectados. En este escenario, cada proyección es un CSAR independiente que contiene los elementos de la topología que se van a desplegar en las distintas plataformas, lo que permite:

- **Reutilizar descripciones originales**. En el CSAR que modela una aplicación se describen todos los componentes que forman parte de ella mediante la metodología especificada en el estándar. Usar un CSAR como contenedor de una proyección permite reutilizar toda la descripción de estos componentes facilitando así el proceso de fragmentación.
- **Composición de proyecciones inmediatas**. Dadas las ventajas expuestas en el punto anterior, la composición de las proyecciones será un proceso de agrupación de componentes de acuerdo a los proveedores finales donde van a ser desplegados y a continuación su empaquetado en un CSAR-proyectado siguiendo las reglas impuestas por el estándar.
- **Soporte del estándar TOSCA**. Toda plataforma que haya adaptado sus servicios a la especificación del estándar cuenta con los mecanismos necesarios para interpretar la descripción de los componentes de una proyección. Además, aprovechando los servicios de los proveedores que implementen el estándar necesarios para ejecutar los planes especificados en un CSAR, aseguramos el soporte al de los planes proyectados para desplegar los componentes de la aplicación.

De acuerdo a estos motivos, este escenario establece un entorno de despliegue de aplicaciones en un contexto de *cloud* heterogéneo siguiendo la metodología propuesta en el estándar de TOSCA, tal y como aparece en la Fig. 4. En esta imagen podemos ver el proceso de fragmentación de un CSAR donde se obtiene una proyección para cada una de las plataformas finales utilizadas, AWS y HP Cloud Computing, siguiendo el código de colores naranja y azul respectivamente. Mediante esta nomenclatura se representa la proyección final de los distintos componentes del CSAR, tanto de la topología como de los planes. Cada una de las proyecciones se representa con un CSAR que contiene todos los componentes necesarios, elementos de la topología y fragmentos del plan de orquestación, las cuales son distribuidas entre las distintas plataformas mediante el entorno de adaptación de TOSCA que estas implementan.

Aplicando esta metodología al ejemplo presentado en la Sección 3.1, la aplicación se modelaría mediante un CSAR. Este sería fragmentado según los proveedo-

res utilizados, en este caso se obtendrían dos proyecciones: una para Amazon con las bases de datos y otra para HP con los módulos webs. En este caso, el nivel de los servicios de despliegue utilizados (IaaS o PaaS) va a depender del soporte proporcionados por la implementación del estándar de los proveedores finales.

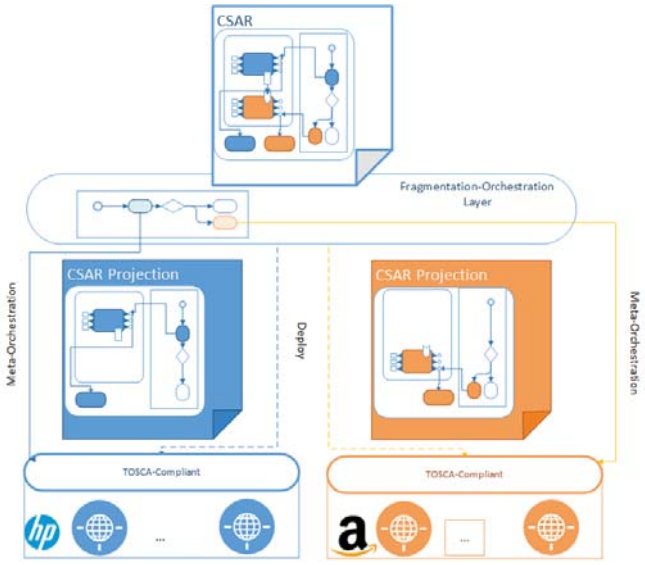


Fig. 4. Proceso de fragmentación, distribución y meta-orquestación de las proyecciones.

En la Fig. 4 observamos una entidad externa a las plataformas *cloud* encargada de la fragmentación de las proyecciones y su distribución a través de los distintos proveedores que denominamos Capa de Fragmentación y Orquestación (*Orchestration-Fragmentation Layer*). El proceso de fragmentación (proyección) de un CSAR no necesita ninguna lógica de adaptación del estándar a ninguna plataforma, no se requiere interpretar el CSAR para la instanciación de la topología sino que solo se va a fragmentar y distribuir las distintas proyecciones a lo largo de las plataformas correspondientes. Esta tarea (fragmentación) es ajena al propio proceso de despliegue de las proyecciones, ya que una vez estas últimas se han distribuido entre los proveedores finales son los entornos de implantación de TOSCA de las plataformas (TOSCA-*compliant*) los encargados de gestionar y desplegar los componentes descritos en dichas proyecciones, obteniendo así un multidespliegue. Consideramos este motivo una base suficiente de la autonomía e independencia de la entidad encargada de esta tarea que, como hemos descrito, es ajena a cualquier plataforma. De esta forma, *(i)* se mantiene el control y la gestión de los componentes desplegados en las distintas plataformas por la capa de adaptación de TOSCA en los proveedores, *(ii)* se mantiene su descripción original que es encapsulada dentro del contenedor especificado por el estándar, y *(iii)* se favorece a la implantación del estándar en nuevas plataformas para

ofrecer sus servicios en despliegues heterogéneos ya que esta capa externa es la encargada de implementar la lógica extra planteada en nuestra propuesta, .

Afrontando estas cuestiones, la capa de fragmentación se encarga de realizar la orquestación entre proveedores, para lo que define un plan de orden superior (*meta-plan*) de orquestación donde se especifica la lectura de estos datos dinámicos y la composición de las relaciones entre los componentes distribuidos. Este meta-plan debe de ser inferido de forma automática del plan original y la topología de la aplicación. Es necesario analizar las relaciones y dependencias entre componentes, comprobando las proyecciones de los mismos para, a continuación, realizar una clasificación exhaustiva de las instrucciones descritas en el plan de orquestación original que operen sobre las relaciones entre componentes con distintas proyecciones. De esta forma, una vez se ha recopilado esta información, se pasa al proceso composición de plan de meta-orquestación extrayendo del plan original los fragmentos necesarios para conectar componentes entre plataformas. Así mismo, podría ser necesario añadirle nuevas instrucciones para completar los posibles huecos que puedan aparecer al conectar secciones del plan original que anteriormente no estaban relacionadas. Con el objetivo de realizar esta tarea, la capa de fragmentación y orquestación requiere *(i)* acceso a los servicios de despliegue de los CSAR en las plataformas finales utilizadas. *(ii)* Estas plataformas deben de ofrecer mecanismos suficientes para realizar la lectura de los datos necesarios y para establecer las conexiones entre los componentes.

3.3. Extensión del estándar

La especificación actual de TOSCA determina un amplio conjunto de características que describen a los componentes de una aplicación basada en el estándar. Sin embargo, a priori, el estándar no facilita ninguna metodología para indicar de forma explícita los proveedores finales donde estos van a ser desplegados, es decir, su proyección. Esto hace imposible determinar el proveedor objetivo de cada uno de los componentes de la aplicación. Es cierto dichos componentes determinan los proveedores donde van a ser desplegados mediante los artefactos de implementación que describen las operaciones y los servicios necesarios en el despliegue y gestión de los componentes de la topología. Aunque en última instancia se podrían usar estos elementos como fuentes de datos para realizar la proyección de un CSAR sobre una plataforma, parece complicado extraer este conocimiento de estos artefactos que pueden ser tan heterogéneos y complejos como el programador que los desarrolla lo desee, por lo que no consideramos que sea una alternativa plausible a las carencias de la especificación en este contexto. Así mismo, estos artefactos no expresan, bajo ninguna circunstancia, la orquestación de los servicios necesarios para obtener un despliegue correcto (información que como hemos visto en el epígrafe anterior es de vital importancia para la composición de las proyecciones). Para hacer frente a esta problemática y facilitar la proyección de un CSAR sobre una plataforma, proponemos una ampliación del estándar con el objetivo de permitir una descripción exhaustiva de los proveedores finales donde van desplegar los módulos de una aplicación.

En la topología de TOSCA los componentes de una aplicación se modelan mediante la plantilla de nodos y relaciones (*Node and Relationship Template*),

ampliando la especificación de estos elementos se pueden definir nuevas propiedades para describir los proveedores finales donde van a ser proyectados. En este sentido, en el Listado 1.1 mostramos nuestra aportación a la especificación de TOSCA[1], donde hemos añadido una cláusula, *projection*, a la definición de los nodos plantilla mencionados para describir las plataformas finales donde van a ser desplegados.

Listado 1.1. Aportación a la definción de TOSCA para especificar los proveedores finales de despliegue.

```
<NodeTemplate id="xs:ID" name="xs:string"? type="xs:QName"
  ...
  <Properties>
            XML fragment
  </Properties>
  <projection provider= "xs:string">
  ...
</NodeTemplate>
```

En el Listado 1.1 podemos ver cómo los valores de las proyecciones son simples cadenas de texto. Con el objetivo de expresar de forma inequívoca las proyecciones de los artefactos es necesario añadir a la especificación identificadores estandarizados que hagan referencia a las plataformas de *cloud* que se van a utilizar, lo que facilitaría el proceso realizado por la capa de fragmentación. Por ejemplo un mapa del tipo clave-valor, como el que se muestra en el Listado 1.2, en el que se establezcan los identificadores de las distintas plataformas. En el ejemplo mostrado en el listado aparecen identificadores para referencia a la plataforma de AWS y HP (usadas en el ejemplo descrito en la Sección 3.1) que, en este caso, se especifican hasta los centros de datos seleccionados en el despliegue. Esta metodología ha sido heredada y adaptada de las tecnologías Brooklyn y Jclouds, las cuales veremos en la Sección 4

Listado 1.2. Ejemplo mapa Clave-Valor para identificar a los proveedores.

```
tosca.location.named.AWS\ Oregon\ = aws-ec2:us-west-2
tosca.location.named.HP\ Cloud\ Arizona-1 = hpcloud-compute
```

En la Sección 2 ya se mencionó que el estándar define un conjunto de buenas prácticas para facilitar su proceso de implantación en una plataforma; sin embargo no se especifican mecanismos de acceso a los datos del despliegue de la topología los cuales, como hemos visto, son necesarios para que la capa de fragmentación pueda llevar a cabo la meta-orquestación de las proyecciones estableciendo las relaciones entre componentes distribuidos en distintos proveedores. Por estos motivos, proponemos ampliar la especificación del estándar definiendo una metodología que permita recolectar datos dinámicos de despliegue y establecer relaciones entre componentes. Dicha metodología podría estar compuesta por un conjunto de servicios REST que especifique cómo una plataforma debe exponer estas funcionalidades desde los mecanismos de implementación del estándar, tal y como

[1] La definición original de estos componentes en está disponible en [6]

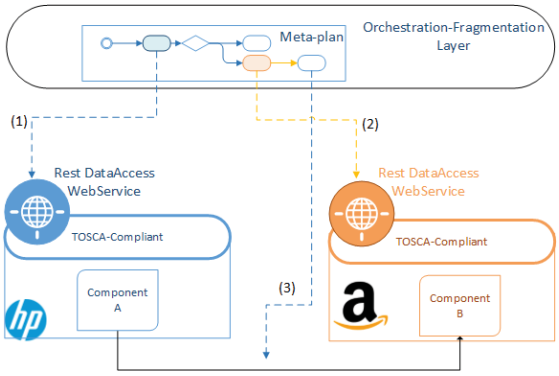


Fig. 5. Plantilla de servicio, topología y demás elementos definidos por TOSCA.

se muestra en la Fig. 5. En esta ejemplifica un proceso de meta-orquestación para conectar dos elementos de una aplicación desplegados en proveedores distintos. Este proceso se compone de tres pasos gestionados por la capa de fragmentación donde se muestra la necesidad de estos mecanismos de comunicación expuestos por las plataformas.

A través de estos servicios la capa de fragmentación se encarga de las siguientes acciones: *(i)* Acceso a los datos del componente A. *(ii)* Envío de datos al componente B. *(iii)* Establecimiento y gestión de la relación entre A y B.

Tras la propuesta que que recoge nuestra idea para llevara acabo despliegues heterogéneos sobre entornos TOSCA-*compliant*, presentamos a continuación el trabajo relacionado.

4. Trabajos relacionados

Existen otras alternativas que ofrecen metodologías para afrontar el desarrollo y el despliegue de aplicaciones en contextos heterogéneos.

La propia organización de OASIS, que ha desarrollado la especificación de TOSCA, propone otro estándar para el despliegue, gestión y monitorización de aplicaciones *cloud* independientemente de la plataforma: CAMP [14] (*Cloud Application Management for Platforms*). En él se define una API común que abstrae a todos los proveedores bajo una misma interfaz, de forma que permite el desarrollo y modelado de aplicaciones apoyadas en esta API, portables a un entorno de plataformas homogéneo. Al igual que TOSCA, CAMP cuenta con una aproximación de implementación estable, Brooklyn (`http://brooklyn.incubator.apache.org/`), un *framework* de control basado en políticas de código abierto para aplicaciones distribuidas que presenta una continua alineación con el estándar, ofreciendo mecanismos para definir aplicaciones de acuerdo a la especificación del estándar, usando la librería (`http://jclouds.apache.org/`) como API. Aunque no deja de ser una aproximación, Brooklyn permite el despliegue de aplicaciones en entornos heterogéneos compatible con todos los proveedores adaptados en Jclouds. No obstante, a diferencia de nuestra propuesta que describe la topología de un sistema, esta herramienta no mantiene una

estructura explícita y expresiva de la aplicación, por lo que una vez realizada la distribución de los elementos es complicado comprobar la estructura de los componentes (cuestión muy importante para controlar el estado la aplicación).

Ubuntu juju (https://juju.ubuntu.com/) también es un *framework* de orquestación que soporta el despliegue heterogéneos de servicios facilitando su interrelación y la descripción topológica, de una forma similar a la de Winery. Sin embargo, aunque está muy extendido en el ámbito comercial, a diferencia de nuestra propuesta no está basado en ningún estándar, lo que desde nuestro punto de vista es esencial de cara a mitigar el *vendor lock-in*.

En la Sección 2 mencionamos OpenTOSCA como una posible aproximación a TOSCA en el que se especifica que la entidad de adaptación de TOSCA debe estar integrada en las plataformas finales, permitiendo así el uso directo de los servicios de los proveedores para el despliegue de las aplicaciones que mantienen un conocimiento exhaustivo de la aplicación; enfoque que mantenemos en nuestra propuesta delegando y confiando el despliegue de las proyecciones en la entidad implementación del estándar por parte de los proveedores. No obstante, OpenTOSCA aunque es un entorno de alta utilidad y usabilidad, es externa a cualquier plataforma. Esta independencia, aunque la consideramos una gran aportación en cuanto a la problemática del *vendor lock-in*, requiere mecanismos externos para el acceso a los servicios de los distintos proveedores (APIs de comunicación), implicando de esta forma una tarea extra de mantenimiento de dichos mecanismos. Por otro lado, a diferencia de nuestra propuesta en la que el conocimiento de los elementos distribuidos en una plataforma se mantenía en la capa de TOSCA-*Compliant*, OpenTOSCA no presenta como objetivo distribuir este conocimiento entre los proveedores finales, ni considera de forma explícita la posibilidad de multi-despliegue en diversos proveedores.

Podemos encontrar otros grupos de investigación y desarrollo que ofrecen sus propias alternativas para solventar los problemas de portabilidad e interoperabilidad y control de plataformas de *cloud*. Por ejemplo, DMTF (*Distributed Management Task Force*) (`http://www.dmtf.org/`) propone *Interoperable Cloud* [15] donde definen varias normas y estándares con el objetivo de definir interfaces entre los clientes y los proveedores de *cloud*, estableciendo así un conjunto de mecanismos que facilitan la portabilidad e interoperabilidad entre las plataformas y los clientes. CIMI (*Cloud Infraestructure Management Interface*) [16] otra posible alternativa desarrollada por CMWG (`http://www.dmtf.org/standards/cmwg`), un subgrupo de DMTF, define la gestión e interacción entre *clouds* a nivel de IaaS estableciendo bajo una API común a todos los proveedores, de la misma forma que pretende hacerlo CAMP. Durante los últimos años tambien han ido ganando importancia los proveedores clasificados como *Cloud Federation*. En términos generales, este tipo de plataformas permiten seleccionar al usuario final los infraestructura sobre la que se quiere realizar un despliegue mediante un conjunto de servicios de terceros. Por ejemplo, Paraiso [17] propone definir un nivel de PaaS sobre el nivel de IaaS de un proveedor.

Aunque estos estándares afrontan el *vendor lock-in*, consideramos que TOSCA cuenta con ciertas ventaja frente a estas especificaciones. TOSCA define un

modelado de los sistemas que van a ser desplegados, que en la mayoría de los casos están formados por un conjunto de componentes heterogéneos interconectados entre ellos. Además, especifica una descripción interoperable de los servicios utilizados mediante un plan de orquestación. De esta forma, no pretende afrontar los distintos problemas de la plataforma mediante la definición e imposición de una API a los distintos proveedores, sino que por el contrario usando este plan define un sistema capaz de usar los servicios que estos exponen. Por otro lado, TOSCA propone una especificación para la topología de una aplicación, describiendo así la estructura de la misma, los elementos que la componen, y cómo se relacionan entre ellos y los distintos servicios de la plataforma. Las ventajas que exhibe TOSCA frente a estos otros enfoques son heredadas por nuestra propuesta.

En este ámbito hemos de mencionar el proyecto SeaClouds (*SEAmless adaptive multi-CLOUD management of service based applicationS* [18]) donde se pretende abordar y apoyar la implementación y administración de aplicaciones complejas multicomponente sobre conjuntos de *clouds* heterogéneos de manera adaptativa. De esta manera se adapta el entorno o contexto de ejecución de la aplicación a las necesidades de la misma, haciendo uso de los estándares y tecnologías que se acaban de describir. Precisamente, la presente propuesta tiene lugar dentro del ciclo de vida del proyecto SeaClouds para abordar cuestiones relacionadas con el despliegue heterogéneo, además de estudiar la alineación con los estándares existentes, previamente mencionados.

5. Conclusiones y trabajo futuro

En este trabajo, hemos propuesto una metodología para realizar el despliegue de aplicaciones en entornos *multi-cloud* tomando como base el estándar de TOSCA, que pretende resolver la problemática de interoperabilidad y portabilidad relacionada con el uso de los servicios de las distintas plataformas de *Cloud Comptuing*.

Nuestra idea sigue la filosofía planteada por el estándar donde las aplicaciones se describen mediante la especificación de: tipos (*Types*) y plantillas (*Templates*) para construir la topología, planes de orquestación para la gestión y control de los mismos y artefactos con los elementos reales que componen el sistema. Una aplicación se distribuirá a través de los distintos proveedores mediante el concepto de *proyección*, que hemos definido como archivos contenedores CSAR, manteniéndonos dentro de la especificación del estándar, y donde serán las capas de adaptación de las plataformas (TOSCA-*compliant*) las encargadas de gestionar el despliegue de todos los componentes proyectadas. Esto permite que, tras un previo estudio, los usuarios puedan optimizar los recursos disponibles diversificando los distintos componentes de una aplicación atendiendo a las características de los servicios ofertados por los distintos proveedores

Esto respeta las aportaciones de TOSCA en cuanto a portabilidad e interoperabilidad entre las plataformas así como todos los esfuerzos realizados para definir una metodología de orquestación de los componentes. La portabilidad y la automatización de despliegue de las aplicaciones que TOSCA describe son cuestiones relevantes que pretenden solventar la problemática del *vendor lock-in*,

características sobre las que hemos pretendido construir nuestra propuesta; dotando así a los usuarios de un entorno de despliegue heterogéneo reutilizable y flexible. Estas cuestiones abren nuevas líneas de investigación sobre aspectos del propio estándar, que constituyen el trabajo futuro que pretendemos abordar:

- **Proyecciones de topología y plan**. Proponemos soluciones para definir la proyección (localización) de los módulos que componen una aplicación de forma inequívoca mediante la adicción de nuevas características a la especificación y la estandarización de los identificadores de las plataformas. Aunque esto posibilita la fragmentación de una topología y los demás componentes, se establecen nuevas líneas de investigación en la descomposición de los planes para orquestar los elementos que componen una proyección. TOSCA describe estos planes como procesos de *workflow*, cuya especificación queda fuera del ámbito del estándar, por lo tanto será necesario un proceso de *matching* entre el plan y la topología de una aplicación para determinar la forma correcta de orquestar los componentes de las proyecciones.
- **Construcción de las proyecciones**. Las proyecciones se encapsulan en CSARs, los cuales ya están especificados en el estándar. Tomando este trabajo como base, el estándar debe de añadir nuevos mecanismos para definir CSAR proyectados sobre las plataformas con el objetivo de optimizar la meta-orquestación de los mismos.
- **Meta-orquestación**. Esta nueva línea de investigación pretende aportar soluciones a las cuestiones relacionadas tanto con el establecimiento de relaciones *inter-clouds* como con los mecanismos necesarios para llevarlas a cabo, lo que implica la ampliación de las capas TOSCA-*compliant* de las plataformas finales.
- **Reconfiguración y variabilidad adaptativa**. Estudio de la posibilidad de abordar la personalización del despliegue heterogéneo utilizando técnicas de variabilidad del software, de forma que los proveedores finales utilizados dependan de un conjunto de restricciones y características identificadas y valoradas por los usuarios finales del sistema.

6. Agradecimientos

Este trabajo ha sido desarrollado con el apoyo de los proyectos: TIN2012-35669, financiado por el Ministerio Español de Economía y Competitividad y FEDER; FP7-610531-SeaClouds, financiado por la Unión Europea; P11-TIC-7659, financiado por la Junta de Andalucía; y la Universidad de Málaga y el Campus de Excelencia Internacional Andalucía Tech.

Referencias

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Communications of the ACM **53** (2010) 50–58
2. Mell, P., Grance, T.: The nist definition of cloud computing (draft). NIST special publication **800** (2011) 7

3. Petcu, D.: Portability and interoperability between clouds: challenges and case study. In: Towards a Service-Based Internet. Springer (2011) 62–74
4. Nicolae, B., Cappello, F., Antoniu, G.: Optimizing multi-deployment on clouds by means of self-adaptive prefetching. In: Euro-Par 2011 Parallel Processing. Springer (2011) 503–513
5. Yang, X., Zhang, H.: Cloud computing and soa convergence research. In: Computational Intelligence and Design (ISCID), 2012 Fifth International Symposium on. Volume 1., IEEE (2012) 330–335
6. OASIS: TOSCA 1.0 (Topology and Orchestration Specification for Cloud Applications), Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.pdf (2012)
7. Lipton, P.: Escaping vendor lock-in with tosca, an emerging cloud standard for portability. CA Labs Research (2012) 49
8. Camara, J., Martin, J.A., Salaun, G., Cubo, J., Ouederni, M., Canal, C., Pimentel, E.: Itaca: An integrated toolbox for the automatic composition and adaptation of web services. In: Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on, IEEE (2009) 627–630
9. OASIS: TOSCA 1.0 (Topology and Orchestration Specification for Cloud Applications), Primer version 1.0. http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.pdf (2013)
10. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: Opentosca–a runtime for tosca-based cloud applications. In: Service-Oriented Computing. Springer (2013) 692–695
11. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery–a modeling tool for tosca-based cloud applications. In: Service-Oriented Computing. Springer (2013) 700–704
12. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Bpmn4tosca: A domain-specific language to model management plans for composite applications. In: Business Process Model and Notation. Springer (2012) 38–52
13. OASIS: Interoperability Demo of OASIS TOSCA. https://www.oasis-open.org/events/cloud/2013/toscademo (2013)
14. OASIS: CAMP 1.0 (Cloud Application Management for Platforms), Version 1.0. http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html/ (2012)
15. DMTF: Interoperable Cloud. http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf (2013)
16. CMWG: CIMI. http://dmtf.org/sites/default/files/standards/documents/DSP0263_1.0.1.pdf (2013)
17. Paraiso, F., Haderer, N., Merle, P., Rouvoy, R., Seinturier, L.: A federated multi-cloud paas infrastructure. In: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, IEEE (2012) 392–399
18. Brogi, A., Ibrahim, A., Soldani, J., Carrasco, J., Cubo, J., Pimentel, E., D'Andria, F.: Seaclouds: a european project on seamless management of multi-cloud applications. ACM SIGSOFT Software Engineering Notes **39** (2014) 1–4

Configurable feature models

Pablo Trinidad*, Antonio Ruiz-Cortés, and Jesús García-Galán

Universidad de Sevilla
{ptrinidad,aruiz,jegalan}@us.es
http://www.isa.us.es

Abstract. Feature models represent all the products that can be built under a variability-intensive system such as a software product line, but they are not fully configurable. There exist no explicit effort in defining configuration models that enable making decisions on attributes and cardinalities in feature models that use these artefacts. In this paper we present configurable feature models as an evolution from feature models that integrate configuration models within, improving the configurability of variability-intensive systems.

Keywords: software product lines, configurable feature models, configuration, model

1 Introduction

Since the set of products that can be manufactured in a Software Product Line (SPL) can be huge, with thousands of products [4, 7] it is necessary to have models that make it possible both to represent the complete set of products in a compact manner and to enable its systematic and automated management. Feature Models (FMs) [3] are one of the most widely used models for this purpose, proposing a compact representation of all the products in an SPL in terms of their features. A feature is a user-visible aspect or characteristic of the domain. Features are connected by means of relationships among them, forming a tree-like structure. Relationships constrain the way in which features can be combined. Besides features, FMs might use cardinalities to group features in the so-called Cardinality-Based Feature Models (CBFMs) and/or attributes to remark non-functional characteristics of products in the so-called Extended Feature Models (EFMs).

The process by which one or more users define the product that best fits their needs by making successive decisions on a particular FM is called *configuration process* [5]. This process is successfully accomplished when a threefold condition fulfils: there are no more user decisions to make, there exists only one product

* This work has been partially supported by the European Commission (FEDER) and Spanish Government under TAPAS project (TIN2012-32273) and IPT-2012-0890-3 (SaaS-FireWall) and the Andalusian Government under COPAS (P12-TIC-1867) and THEOS (TIC-5906) projects

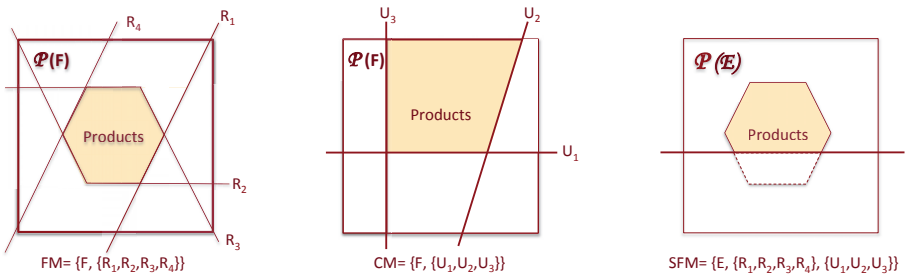
in the SPL satisfying each and every user decision, and there are no contradictions among decisions. The decisions made by users in a configuration process are collected by the so-called Configuration Model (CM), or simply a *configuration*. CMs have not been specifically studied in SPLs, being widely accepted the three-set model that represents user decisions in terms of selected, removed and undecided features. We have found three limitations in CMs that we aim to save in this paper.

First, despite cardinalities and attributes are used in Extended and Cardinality-Based Feature Models (ECBFMs) to represent relevant information, current CMs are unable to represent decisions on them, only enabling the decision making on features. It impedes a user making decisions about attributes such as 'I want a Smart Home System (SHS) that costs less than 2.000 €', or about cardinalities such as 'I want a SHS with two Internet connections'. We affirm that current FMs are not *fully-configurable* since CMs are unable to make decisions on any element of the FM.

Second, a strong relationship between elements in the FM and in the CM arises when attributes and cardinalities come into play. A CM must ensure that a user decision on an attribute is made within the attribute domain, which is specified in the FM. The same happens for cardinalities, whose valid values for user decisions are modelled in the FM. FMs and CMs share elements and the values those elements can take in a user decision. There are two main approaches to deal with information sharing: keeping two separate models in constant synchronisation, communicating any change in the FM to CMs, or combining FMs and CMs in a unique model around the common elements.

Third, some authors [2, 9] have used annotations on FMs to depict decisions on features. But there is no graphical representation that supports attributes and cardinalities.

From these problems we consider that it is possible to extend CMs to enable fully-configurable ECBFMs. We propose Configurable Feature Models (CFMs) [8] as a kind of model that *(i)* combines FMs and CMs in a unique model, enabling the representation of user decisions on attributes and cardinalities; *(ii)* takes advantage of the information that FMs and CMs share; and *(iii)* represents CMs together with FMs in a backwards-compatible graphical notation. In Section 2, we interpret FMs and CMs as a set of constraints on the same set of elements. In Section 3 we describe the main concepts in CFMs. In Section 4 we propose an abstract model for CFMs. Two different representations of CFMs are presented, each of them for a different purpose. In Section 5, we firstly present the configurable feature diagrams as a graphical representation of CFMs. in Section 6, a UML configurable feature metamodel is proposed as a formalisation of CFMs that enable their transformation to other metamodels. Last, Section 7 presents the next steps to make the most of CFMs.



(a) Feature model (b) Configuration model (c) Configurable feature model

Fig. 1. A visual metaphore of FMs, CMs and CFMs

2 Rationale

A FM describes a set of products in terms of features and relationships. A product is a subset of features that satisfy all the relationships. From these definitions, we can interpret a FM in terms of the following sets. Let F be the set of features in a SPL, and let $\mathcal{P}(F)$ be the set of all the possible combination of features. The relationships can be interpreted as constraints on $\mathcal{P}(F)$. These constraints define the set of products (P) as a subset $P \subseteq \mathcal{P}(F)$.

CMs represent the user decisions in terms of three sets of selected, removed and undecided features. User decisions can be interpreted as another different way to constrain the set of all the possible combination of features $\mathcal{P}(F)$ with a different kind of constraints. Therefore FMs and CMs can be regarded as two different models that describe subsets of $\mathcal{P}(F)$.

This vision also fits into CBFMs and EFMs where cardinalities and attributes are added to the set of features. In this case, a new set that represents all the features, cardinalities and attributes in an ECBFM is necessary. This set is denoted as the set of elements E. In this case, relationships can be interpreted as constraints on $\mathcal{P}(E)$, i.e. the space of all the possible combinations of elements. Symmetrically, user decisions in CMs could also be interpreted as constraint on $\mathcal{P}(E)$ if they were fully-configurable, but current CMs keep on interpreting the CM as a subset of $\mathcal{P}(F)$.

In order to enable fully-configurable FMs, we propose extending CMs to support cardinalities and attributes besides features. The resulting models must also satisfy three main requirements in order to provide fully-configurable FMs: *(i)* user decisions can only refer to elements in E so it must avoid any reference to invalid cardinalities or attribute values; *(ii)* CMs must allow multiple users making decisions at the same time, and *(iii)* CMs must distinguish between user and automatic decisions.

As a solution, we propose incorporating FMs and CMs together in a single model that we have coined as Configurable Feature Model (CFM). The resulting model stores together the set of elements E, and two sets of constraints, one for

relationships and another one for user decisions (see Figure 1). A CFM supports multiuser configurations and distinguishes between user and automatic decisions. CFMs are the first fully-configurable FMs, able to represent user decisions on attributes and cardinalities. Table 1 compares the capability of dealing with features, cardinalities and attributes in CFMs, basic FMs, CBFMs, EFMs and ECBFMs. In next Section we propose an abstract model for CFMs and the relevant concepts that arise from their use.

Table 1. A comparison of the use of elements in different kinds of feature models

Kind of constraint	Basic FM	CBFM	EFM	ECBFM	CFM
Relationship	F	F, C	F, A	F, C, A	F, C, A
User decisions	F	F	F	F	F, C, A

F = Features, C = Cardinals, A = Attributes

3 Main concepts

A CFM contains information about the elements in a SPL, which are features, cardinalities and attributes, hitherto stored in FMs. Depending on the decisions a user can make on elements, each element has its own set of states. So features have a selected or removed state; Cardinalities have as many states as cardinals in its range; Attributes have as many states as values they can take. An assignment of states for every element in the CFM defines the characteristics of what is called a *potential product*.

To determine which of the potential products can be built within the SPL, a set of *relationships* constrains the valid combinations of states, defining what is called the set of *SPL products*. A CFM also collects all user decisions, which are constraints that define a subset of potential products named *configuration*. The set of SPL products that are also found in the configuration is called the set of *compatible products*, and brings together the SPL products that meet all user decisions.

CFMs distinguish among user and automatic decisions. An automatic decisions is the one that is made by an Automated Analysis of Feature Models (AAFM) operation to support the configuration process. An automatic decision can be defined as the one that if a user makes it, the set of compatible products remains unaltered. They are computed and stored separately from user decisions unless a user explicitly makes such a decision.

Figure 2 proposes a geometrical interpretation of the concepts presented in this Section.

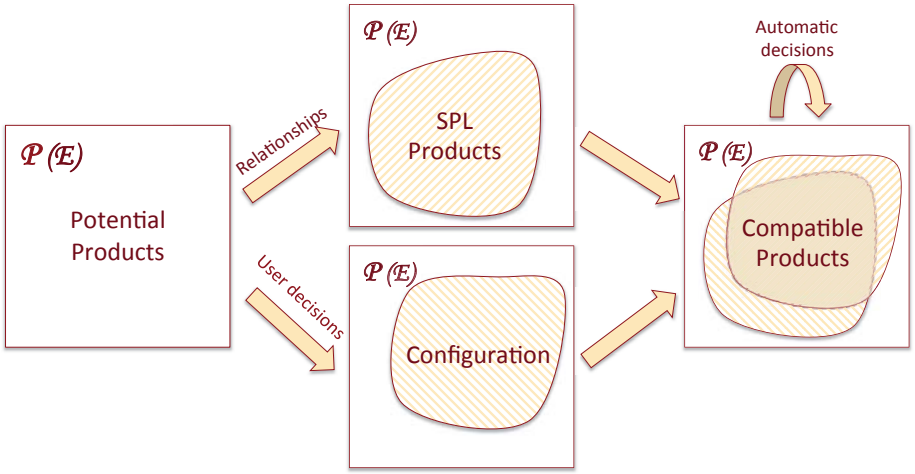


Fig. 2. A geometrical interpretation of basic concepts in CFMs

4 Abstract model for CFMs

In this Section we give a transformational semantics to CFMs in terms of set theory, relying on a new vision of products as an assignment of states to elements. In FMs, each product is described as a different subset of features. In CFMs, all the products share the same set of elements, denoted by a non-empty set $E = \{E_1, ..., E_n\}$ where each E_i is an element in that model, either features, cardinalities or attributes. A product is defined as an assignment of states to every element in E such that each element has a set of available states that depends on the kind of element. This way, features have selected or removed states to indicate their presence or absence in a product; a cardinality has a cardinal value as state to indicate the number of features that are in a selected state within a set relationship; attributes have a state which corresponds to a value that represents the quality or behaviour of their linked features.

To define which states every element can have, a CFM has a set of available state sets $AS_1, ..., AS_n$ such that AS_j is the set of available states for an element E_j. Let us consider an SPL with 6 elements $E = \{F_R, F_A, F_B, F_C, C_1, Mem\}$ as an example. F_R is the root feature; F_A, F_B and F_C are child features; C_1 is a cardinal affecting the 3 child features; Mem is an attribute linked to the root feature. The available states for each elements could be: $AS_{F_R} = \{sel\}$, $AS_{F_A} = AS_{F_B} = AS_{F_C} = \{sel, rem\}$, $AS_{C_1} = \{1, 2, 3\}$, $AS_{Mem} = \{128, 256\}$. This way, the root feature F_R must have a selected state, features F_A, F_B and F_C can be either in a selected or removed state, C_1 cardinality, which affects features F_A, F_B and F_C, must be either 1, 2 or 3 and the product can have a 128 or 256 kilobytes memory size. We define the set of potential products as follows:

Definition 1 (Set of potential products). *Let AS_j be the set of available states for the jth element $e_j \in E$. The set of all the potential products in a CFM is defined by the cartesian product $D = AS_1 \times ... \times AS_n$. A* potential product *corresponds to any tuple $(s_1, ..., s_n) \in D$.*

In the previous example, the set of potential products D is comprised of 32 different potential products. $S_1 = (sel, sel, sel, sel, 2, 256)$ and $S_2 = (sel, rem, sel, rem, 1, 128)$ are examples of potential products. However, not all of them are *SPL products*, i.e. products that effectively can be built in the SPL. To define the set of all the SPL products, a first option that consists of enumerating all of them is immediately discarded when the number of SPL products shoots up. A set in general can be defined by a list of its elements or by a subset that satisfies a condition or constraint. CFMs use constraints to define the set of SPL products as a subset of the set D of potential products. A CFM has a set of *relationships* $R = \{R_1, ..., R_i\}$ that establishes the conditions a potential product must necessarily fulfill to be an SPL product. So the set of SPL products is defined as follows:

Definition 2 (Set of SPL products). *Let $R = \{R_1, ..., R_i\}$ be the set of relationships in a CFM and D its set of potential products. The set of SPL products defined by the CFM is:*

$$P = \{(s_1, ..., s_n) \in D | R_1 \wedge ... \wedge R_i\}$$

It might be the case that there exist no product satisfying all the relationships and therefore $P = \emptyset$. In this case, it is said that the CFM is *void* or *invalid*.

The kinds of constraint that we propose for CFMs are the same than those used for FMs. Table 2 shows a list of constraints that can be used to represent the relationships in a CFM. So for example, if `R` is a parent feature linked by a set relationship with three child features `A`, `B` and `C` and affected by cardinality C_1, the following constraint and set of SPL products can be defined:

$$P = \{(s_R, s_A, s_B, s_C, s_{C_1}, s_{Mem}) | set_3(s_R, s_{C_1}, s_A, s_B, s_C)\}$$

A CFM also stores a configuration which comprises all the decisions made by one or more users in a given moment. Decisions are collected in any order, supporting a parallel configuration process. These user decisions set a partition on the set of potential products D in a SPL: those that satisfy user decisions and those that do not. Following this criterion, we define a configuration as follows:

Definition 3 (Configuration). *Let $U = \{U_1, ..., U_j\}$ be a set of constraints defined on D, describing the user decisions in a CFM. A configuration C_U is defined as the set of potential products that satisfy all the user decisions:*

$$C_U = \{(s_1, ..., s_n) \in D | U_1 \wedge ... \wedge U_j\}$$

Table 2. Kinds of relationship constraints in a CFM

$mandatory(s_p, s_c)$	$\equiv s_p = s_c$
$optional(s_p, s_c)$	$\equiv s_c = sel \Rightarrow s_p = sel \wedge s_p = rem \Rightarrow s_c = rem$
$set_2(s_p, s_c, s_{c_1}, s_{c_2})^\dagger$	$\equiv s_p = sel \Leftrightarrow numSelChild_2(s_c, s_{c_1}, s_{c_2}) \wedge$ $s_p = rem \Rightarrow s_{c_1} = rem \wedge s_{c_2} = rem$
$set_3(s_p, s_c, s_{c_1}, s_{c_2}, s_{c_3})^\dagger$	$\equiv s_p = sel \Leftrightarrow numSelChild_3(s_c, s_{c_1}, s_{c_2}, s_{c_3}) \wedge$ $s_p = rem \Rightarrow s_{c_1} = rem \wedge s_{c_2} = rem \wedge s_{c_3} = rem$
...	...
$depends(s_1, s_2)$	$\equiv s_1 = sel \Rightarrow s_2 = sel$
$excludes(s_1, s_2)$	$\equiv \neg(s_1 = sel \wedge s_2 = sel)$

$^\dagger numSelChild_i(s_c, s_{c_1}, ..., s_{c_n})$ is a predicate that is true whenever the number of selected states in $s_{c_1}, ..., s_{c_n}$ coincides the cardinal s_c.

Users can make two kinds of decisions as shown in Table 3. A user can either make a *choose decision*, that assigns a state for one element, or a *discard decision* that avoids an element having a certain state. So for example, $U_1 = choose(s_B, sel)$ identifies a user selecting feature B; $U_2 = discard(s_{Mem}, 128)$ identifies a user refusing a 128Kb bandwidth. With these two kinds of decisions, users delimit the set of potential products by means of successive refinements.

In case that user decisions contradict each other, as two users choosing selected and removed states for the same feature for example, then the set $C_U = \emptyset$. This situation is known as a *contradictory configuration*.

Table 3. Kinds of decision constraints in a CFM

$choose(s_e, S) \equiv s_e = S$
$discard(s_e, S) \equiv s_e <> S$

From the configuration and the set of SPL products, it can be determined the set of *compatible products*, i.e. the subset of SPL products that satisfy the criteria established by user decisions. The set of compatible products is defined as follows:

Definition 4 (Set of compatible products). *Let P be the set of SPL products defined by the relationship constraints $R_1, ..., R_i$, and let C_U be a configuration defined by the user decision constraints $U_1, ..., U_j$. The set of compatible products (CP) is defined as:*

$$CP = P \cap C_U = \{(s_1, ..., s_n) \in D | R_1 \wedge ... \wedge R_i \wedge U_1 \wedge ... \wedge U_j\}$$

With the above definitions, we are able to represent a CFM in a compact manner as follows:

Definition 5 (Configurable Feature Model). *A CFM can be represented by a 4- tuple* (E, D, R, U) *such that* E *is the set of elements in the CFM,* D *is the set of potential products,* R *is the set of relationships and* U *is the set of user decisions.*

From the information contained in this tuple, the set of potential, SPL and compatible products can be deduced. Next we introduce three concepts that can be defined on top of CFMs: CFM states, element states and automatic decisions.

4.1 CFM states

From the set of compatible products (CP), it is possible to identify singular situations or *CFM states*:

- Initial state: a CFM is in its initial state when there exist no user decisions, i.e. $U = \emptyset$. In this case, the configuration coincides with the space of potential states ($C_U = D$) and therefore the set of compatible products coincides the set of products, i.e. $CP = P$.
- Final state: a CFM is in a final state when it only determines one compatible product, i.e. $|CP| = 1$. When a final state is reached, it can be affirmed that a product satisfying all the user decisions has been found. The final adjective comes from the inability of the CP set to evolve in a manner that the set of compatible products is reduced even more.
- Intermediate state: A state is intermediate if $U \neq \emptyset$ and $|CP| > 1$. This is the most common state in a CFM and it corresponds to a situation in which users may still made decisions.
- Invalid state: if $CP = \emptyset$ at any moment, a CFM is said to reach an invalid state. This state can be reached because the set of CFM products is empty ($P = \emptyset$), or the configuration is contradictory ($C_U = \emptyset$), or there exist no compatible product satisfying the user decisions ($P \cap C_U = \emptyset$).

A CFM state makes reference to the instant in which the configuration process is. This way, the configuration process starts with a CFM in its initial state. User decisions provoke the CFM state to change to an intermediate state. A final state can be reached if there exist only one product satisfying all the user decisions. In case a configuration defines no compatible product, an invalid state is reached.

4.2 Element states and automatic decisions

From a CFM, it is possible to extract relevant information about the elements and states on which users might still make decisions without reaching an invalid state. For this purpose, we define the concept of *element state* as the set of all the states an element has for every compatible product in CP.

Definition 6 (Element State). *Let E_i be an element in a CFM and CP the set of compatible products. The element state for that element E_i is defined as follows:*

$$State(E_i) = \bigcup_{s \in CP} \pi_i(s)$$

Being $\pi_i(s)$ the projection function that extracts the ith element from a tuple s that represents a compatible product.

So for example, let us consider the following set of compatible products:

$$\begin{aligned} E &= \{F_R, F_A, F_B, F_C, C_1, Mem\} \\ CP &= \{P_1, P_2, P_3\} \\ P_1 &= \{sel, rem, sel, sel, 2, 256\} \\ P_2 &= \{sel, rem, rem, sel, 1, 128\} \\ P_3 &= \{sel, rem, sel, sel, 2, 128\} \end{aligned}$$

The following element states are obtained from these compatible products:

$$\begin{aligned} &State(F_R) = \{sel\},\ State(F_A) = \{rem\},\ State(F_B) = \{sel, rem\}, \\ &State(F_C) = \{sel\},\ State(C_1) = \{1, 2, \},\ State(Mem) = \{128, 256\} \end{aligned}$$

An element state is used to infer what we call the set of *automatic decisions* (A). An automatic decision is the one that if it is made by a user, the set of compatible products remains unaltered. This way, if $A = \{A_1, ..., A_k\}$ represents the set of automatic decisions, and being C_A the subset of all the potential states that satisfy such automatic decisions:

$$C_A = \{(s_1, ..., s_n) \in D | A_1 \wedge ... \wedge A_i\}$$

it is possible to affirm that

$$CP \cap C_A = CP \Rightarrow CP \subseteq C_A$$

This set of automatic decisions is used to assist users in the decision making process, avoiding as far as possible to reach for invalid states. Automatic decisions can be calculated from the element states. Let E_i be an element in a CFM having AS_i as available states and $State(E_i)$ as its corresponding element state. Depending on the element state, an automatic decision can or cannot be inferred as follows:

- $State(E_i) = AS_i$: in this case, the element is said to be in an *undecided* state. It means that no user has made any decision that affects it and no automatic decision can be made on it.
- $|State(E_i)| = 1$: in this case, it can be interpreted that an automatic decision has been made on E_i to choose the unique state in $State(E_i)$
- $|AS_i| > |State(E_i)| > 1$: in this case, it can be interpreted that all the states that are not in $State(E_i)$ have been discarded. It means that those states in the set $AS_i - State(E_i)$ can be automatically discarded.

– $State(E_i) = \emptyset$: this situation can only be given if $CP = \emptyset$ in which case it does not make sense to calculate any automatic decision since there is no compatible product.

Each automatic decisions obtained following the above criteria must be checked if it already exists as a user decision, in which case they are disposed. For the previous example, the following set of automatic decisions can be inferred:

$$A = \{choose(s_A, sel), choose(s_C, sel), discard(s_{C_1}, 3)\}$$

5 Configurable feature diagrams

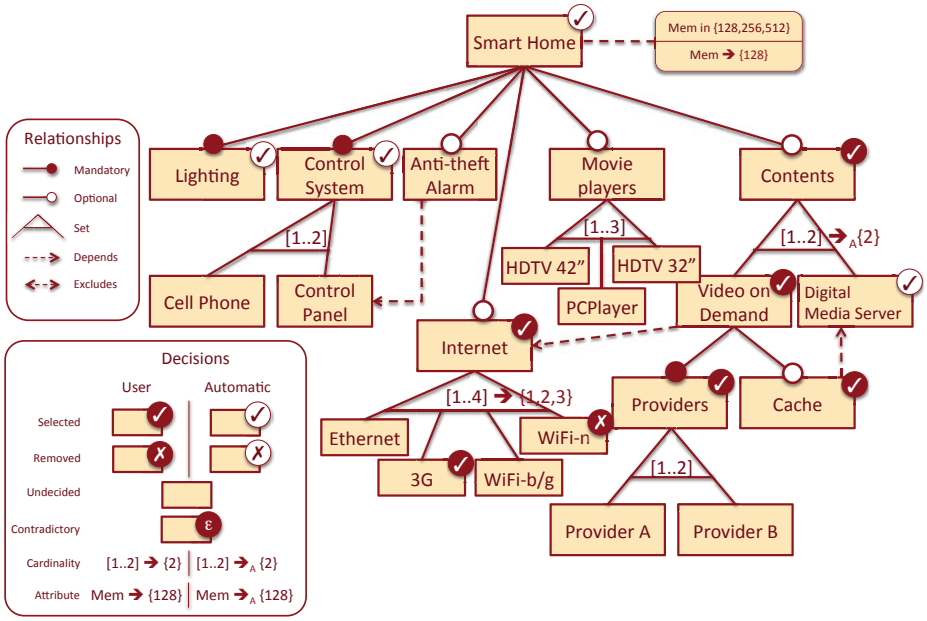


Fig. 3. An example of a Configurable Feature Diagram

Configurable feature diagrams are a graphical notation of CFMs based on feature diagrams [6]. The representation of elements and relationship constraints is the same that FMs: a tree-like structure where features are boxes linked by different kinds of lines that represent the relationships among features. Cardinalities are also drawn together with the corresponding set relationship.

User and automatic decisions are not explicitly depicted in the diagram. Instead of it, element states are drawn. If a feature state is $\{sel\}$, it is drawn as a tick (✓) within a circle in a corner of the feature box; if a feature state is

$\{rem\}$, a cross (✗) is used instead; if a feature is undecided, no specific mark is assigned; if two or more users have made contradictory decisions on a feature, it is marked as ε. It the feature state is inferred by user decisions, then a filled circle is used; if it can only be inferred by automatic decisions, a non-filled circle is used instead.

Decisions on cardinalities and attributes are drawn as $E_d \rightarrow E_s$ where E_d is the cardinality or attribute domain and E_s is the element state for that cardinality or attribute. For example a cardinality [1..3] whose state has been set to $\{2\}$ is drawn as $[1..3] \rightarrow \{2\}$. If state 2 is removed but no other state has been chosen, it is represented as $[1..3] \rightarrow \{1, 3\}$. Likewise, memory attribute (Mem) has three available states $AS_{Mem} = \{128, 256, 512\}$. If its state is set to 128 Mb, it is drawn as $Mem \rightarrow \{128\}$. If a user discards needing 512 Mb, then it is depicted as $Mem \rightarrow \{128, 256\}$. Just in case any state is inferred only from automatic decisions, $E_d \rightarrow_A E_s$ is used to remark that a state has changed due to automatic decisions. Figure 3 shows an example of a configurable feature diagram.

6 Configurable feature metamodel

A metamodel is a rigorous definition of a model that describes their main concepts, relationships and rules. We propose the Configurable Feature Metamodel (CFMM) as a rigorous definition of CFMs. Besides rigour, metamodels can be used as development artefacts in Model-driven Engineering (MDE). It enables to build analysis tools by the definition of transformations into other declarative languages which can be used to reason about them.

Obtaining a metamodel is a design exercise. So we firstly propose a list of objectives the design must satisfy according to the abstract model presented in Section 4:

Objective 1. Representing a set of elements, each of which has a set of available states.

Objective 2. Representing the set of SPL products by means of a set of relationship constraints.

Objective 3. Representing decisions as a set of constraints, distinguishing among user and automatic decisions.

Objective 4. Enabling the computation of element states according to user and automatic decisions.

Objective 5. Proposing an extensible metamodel that supports the future addition of new kinds of relationships and elements.

Figure 4 depicts our proposal of a CFMM using a UML class diagram that satisfies the above objectives. A CFM is represented by a `ConfigurableFeatureModel` instance. It is a container of `Element`, `Relationship` and `Configuration` instances. The elements in a CFM can be either `GenericFeature`, `Cardinality` or `Attribute` instances. There exists a class for each kind of relationship such as mandatory, optional, set, requires, excludes and attribute relationships. A

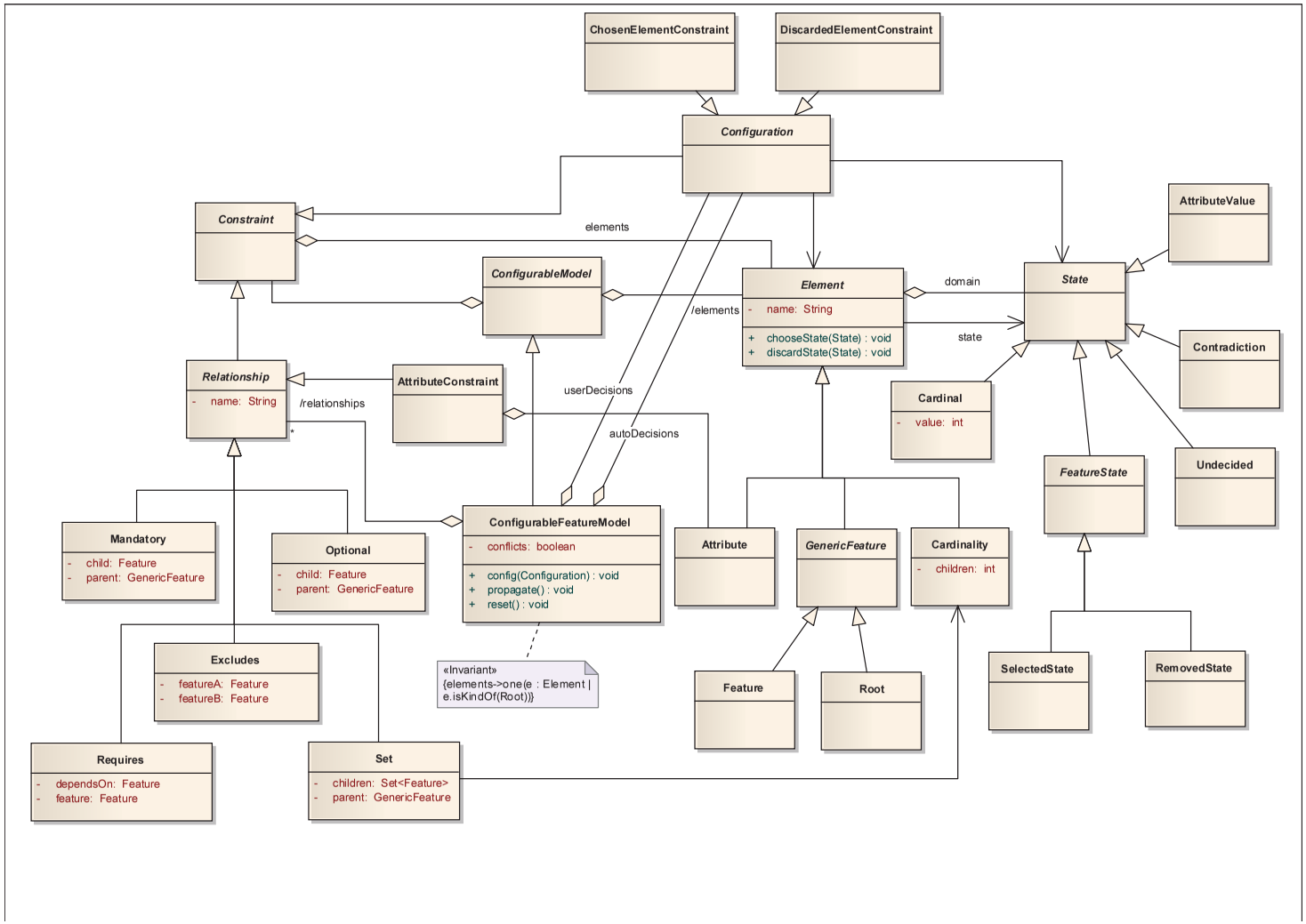


Fig. 4. Configurable Feature Metamodel in UML format

`GenericFeature` class is defined to distinguish between the root and remaining features. It is used to avoid the incorrect use of the root feature in relationships such as the root feature being the child of a relationship or an exclusion between a root and any other feature that provokes a dead feature.

The decisions can be defined in terms of an element selection (`ChosenElementConstraint` class) or discarding (`DiscardedElementConstraint` class). A CFM distinguishes between user and automatic decisions, storing them in two different aggregates. Decisions are introduced in the model by means of the model edition operations shown in the Figure within the `ConfigurableFeatureModel` class. Remaining operations, such as those to build a CFM from scratch or to modify elements and relationships have been left out of the Figure since they are not relevant for the purpose of this paper.

Every element has a domain and a current state. The domain is the set of available states. Each kind of element has its own element-specific available states: `SelectedState` and `RemovedState` for features, `Cardinal` for cardinalities and `AttributeValues` for attributes. An element state is immediately updated when a user decision is introduced into a CFM. Initially, every element state contains an `Undecided` state instance to indicate that no decision has been made on them. As decisions are added, the state of the affected elements changes to represent the decisions. In case a contradiction is found among decisions, such as a feature that is selected and removed at the same time, the `Contradiction` state instance is set as the state for the affected elements.

Last, there exist some generalisation points to increase the extensibility of the metamodel. A `ConfigurableModel` class is defined in case the constraint-element-state structure wants to be reused for other models; relationships and decisions are considered as two different kinds of `Constraint`. In case we want to add a new kind of constraint that is not covered by relationships and decisions, it can be done as an extension of this class.

7 Conclusions and further work

In this paper we have presented CFMs as a new kind of model that enables fully-configurable FMs. CFMs incorporate the concept of element states to describe the dynamic behaviour of CFMs which is a novel approach in the description of products in a SPL. CFMs distinguish among potential products as any combination of element states, SPL products as those potential products that satisfy the relationship contraints, and compatible products as the SPL products that satisfy the configuration constraints. They also allow to work with multiple users making decisions on a CFM at the same time and distinguishes between user and automatic decisions.

Configurable feature diagrams are proposed to graphically describe CFMs. They are inspired in feature diagrams where states are added to represent user and automatic decisions.

Besides the rigorous definitions given in this paper for CFMs, we define them in terms of an UML metamodel. Metamodels enable their use in MDE tools. The

CFMM takes extensibility as a major concern. The extension mechanisms allow the future exploration of the impact of continuous attributes which have been left out of this proposal. The metamodel presented in this paper will play a key role in the implementation of analysis operations that enable the extraction of relevant information from CFMs such as counting products, validating a configuration, detecting and explaining errors, explaining invalid configurations, etc. Our future work will consist in the proposal of a catalogue of analysis operations inspired in the AAFM [1] that introduces new operations where attributes and cardinalities are first-level artifacts.

References

[1] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 9 2010.

[2] C. Cetina, P. Trinidad, V. . Pelechano, and A. Ruiz-Cortés. Customisation along lifecycle of autonomic homes. In *3rd International Workshop on Dynamic Software Product Line (DSPL09)*, 2009.

[3] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Nov. 1990.

[4] F. Lösch. *Optimization of variability in software product lines: a semi-automatic method for visualization, analysis, and restructuring of variability in software product lines.* PhD thesis, University of Stuttgart, 2008.

[5] R. Rabiser, P. Grunbacher, and D. Dhungana. Supporting product derivation by adapting and augmenting variability models. In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 141–150, Sept 2007.

[6] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Feature Diagrams: A Survey and A Formal Semantics. In *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, Minneapolis, Minnesota, USA, Sept. 2006.

[7] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber. Introducing pla at bosch gasoline systems: Experiences and practices. In R. L. Nord, editor, *Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings*, volume 3154 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2004.

[8] P. Trinidad, A. Ruiz-Cortés, and D. Benavides. *Automated Analysis of Stateful Feature Models*, chapter 30, pages 375–380. Springer Berlin Heidelberg, 2013.

[9] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty, and A. Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094 – 1107, 2010.

Un Enfoque Basado en Modelos para incorporar Requisitos No Funcionales y de Integración de Software en el Diseño de Arquitecturas Orientadas a Servicios

M. Guessous[1,2], M.A. Barcelona[1,2], L. García-Borgoñón[1,2], and M. Alba[2]

[1] Instituto Tecnológico de Aragón, c/ María de Luna 7, 50018 Zaragoza,
mguessous@ita.es, mabarcelona@ita.es, laurag@ita.es
[2] Universidad de Sevilla, Av. Reina Mercedes S/N, 41012 Sevilla,
miguel.barcelona@iwt2.org, laura.garcia@iwt2.org, manuel.alba@iwt2.org

Resumen El interés creciente por parte de las organizaciones de desarrollo de software para disponer de metodologías y herramientas para aumentar la calidad de los productos y reducir el tiempo de desarrollo, exige introducir mejoras continuas que permitan agilizar y automatizar partes de los procesos. *Navigational Development Techniques* (NDT) es una metodología que cubre todas las fases del proceso de desarrollo y que ha tenido una gran aplicación práctica en la industria en los últimos años. Este trabajo pretende realizar una propuesta basada en modelos para definir requisitos no funcionales (RNF) y de integración de software, en especial los que pueden afectar al diseño de la arquitectura de sistemas distribuidos, como son la eficiencia y la escalabilidad, y establecer mecanismos de transformación para que puedan ser trazados y considerados en el diseño arquitectónico de un sistema dentro de NDT. En este trabajo emergente se presenta el problema a resolver, el estado del arte, la metodología que vamos a seguir y los principales resultados que esperamos obtener con nuestra investigación.

Palabras Clave: Requisitos No Funcionales, Navigational Development Techniques, Arquitecturas Orientadas a Servicio, Enfoque Basado en Modelos, *Model Driven Engineering*, Integración de componentes software

1 Introducción

El interés creciente por parte de las organizaciones de desarrollo de software para disponer de metodologías y herramientas para aumentar la calidad de los productos y reducir el tiempo de desarrollo, exige introducir mejoras continuas que permitan agilizar y automatizar partes de los procesos. *Navigational Development Techniques* (NDT) [1] es una metodología que ofrece una suite de herramientas que cubren todo el ciclo de vida de software y que, tomando como base el paradigma guiado por modelos, define metamodelos que representan las

diferentes fases del ciclo de vida. NDT incorpora un conjunto de reglas de tranformación entre modelos (M2M, Model-To-Model) y desde los mismos hasta el código (M2T, Model-To-Text), permitiendo igualmente integrar nuevos conceptos definidos mediante metamodelos e integrarlos en su entorno metodológico con estas reglas de transformación. Este trabajo pretende realizar una propuesta basada en modelos para definir requisitos no funcionales (RNF) y de integración de software, en especial los que pueden afectar al diseño de la arquitectura de sistemas distribuidos, como son la eficiencia y la escalabilidad, y establecer mecanismos de transformación para que puedan ser trazados y considerados en el diseño arquitectónico de un sistema dentro de NDT. La figura 1 muestra una visión general de las distintas fases del ciclo de vida de software (en la parte izquierda de la imagen), se ha tenido en cuenta un ciclo de vida secuencial para facilitar la descripción de la figura, aunque NDT en realidad puede trabajar con ciclos de vida de otro tipo (basado en prototipos, etc.), y los metamodelos de RNF y de integración de software así que sus transformaciones en metamodelos y en código (en la parte derecha de la imagen) que este trabajo pretende definir y describir. El resto del artículo se estructura como sigue. En la sección 2 se pre-

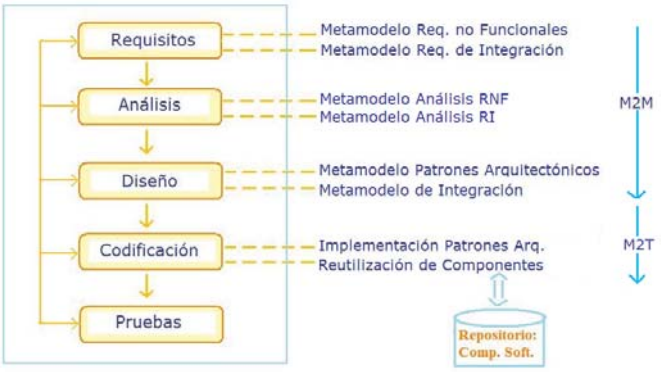


Fig. 1. Vision general del ciclo de vida de software y los metamodelos de RNF y de integración de software

senta un análisis del estado del arte en la temática. Seguidamente el conjunto de actuaciones previstas se detallan en el apartado 3. Por último, los resultados esperados se enumeran en la sección 4.

2 Trabajo Relacionado

Desde hace tiempo se empezó a hablar de la siguiente era del diseño orientado a objetos y su implementación manual que será apoyada sobre la reutilización disciplinada y estratégica del software (*Product Line Software*) y la generalización

de los componentes[2] permitiendo la generación automática de código a partir del modelo conceptual. Por su parte Jan Bosch enumeró en su libro *Design and Use of Software Architectures*[3] los desafíos que se tiene que afrontar la ingeniería del software y que consisten en identificar la manera que permite reducir drásticamente el coste de desarrollo y el *time-to-market* de los productos software y mejorar los atributos de calidad (RNF) de estos productos. A raíz de esto se han llevado a cabo varios trabajos que concentraron en estudiar los requisitos no funcionales que los sistemas deben satisfacer además de los requisitos funcionales[4] y como se traducen estos requisitos en la fase de diseño en modelos arquitectónicos[5–8].

En el marco del desarrollo de lineas de producto, el trabajo[9] utiliza MDD para dar soporte sistemático y automatizado al proceso de derivación y validación de arquitecturas software conducido por los RNF de manera que mejora la calidad de los productos obtenidos a partir de dichas arquitecturas. La iniciativa más similar a la presente propuesta que versa sobre el metamodelado de los RNF y sus transformaciones en la distintas fases del desarrollo, es el trabajo citado en[10] que presenta una extensión del lenguaje de modelado *Systems Modeling Language* (SysML) para que pueda soportar la descripción y validación de RNF, y proponiendo en concreto el metamodelado del requisito de alto rendimiento y un algoritmo que valida el cumplimiento de este requisito en el sistema desarrollado. Tanto este trabajo como los trabajos[11, 12] introducen metamodelos y sus transformaciones en la distintas fases de desarrollo sin llegar a la generación automática de código. Con este trabajo intentamos dar un paso más ofreciendo no solo el soporte y la documentación de los RNF en las fases de requisitos-análisis-diseño sino permitir la generación automática del código de los patrones arquitectónicos (sistemas distribuidos en primer lugar) lo que permite agilizar el proceso de desarrollo de software.

Por otro lado, la reutilización e integración de módulos y componentes software ha sido también un tema de interés para la comunidad investigadora. Prieto-Diaz et al.[13] especifican tres etapas para reutilizar software: i) acceder al código; ii) entender ese código, y depende tanto de la experiencia del desarrollador como de las características propias del programa como el tamaño, la complejidad, la documentación y el lenguaje de programación, y; iii) adaptar el código, que depende de la diferencia entre los requerimientos del componente existente y las habilidades del reutilizador. Meyer[14] menciona la naturaleza repetetiva en los programas informáticos y que solo el 15% del código desarrollado es de propósito original. Gill[15] destaca el nuevo rol de los componentes software dentro de las organizaciones de desarrollo, donde los componentes dejan de ser construidos para un sistema en concreto, pasando a ser integrados en varios sistemas y expresa con Yacoub et al.[16] las características de estos componentes desde una clasificación en tres categorías: i) la descripción informal, considerando la edad (cuando más se reutiliza el componente desde su primer uso más madurez obtendrá), la fase o el nivel donde va a ser reutilizado o el problema que el componente resuelve; ii) sus interacciones externas con otras aplicaciones y con la plataforma donde reside, y entre sus propiedades: interop-

erabilidad, portabilidad, tecnología, características no funcionales, etc, y; iii) sus aspectos internos, relacionados con la codificación y ejecución del componente. La evaluación del grado de reutilización de un componente software es otro aspecto que ha sido estudiado por los investigadores como podemos probar en los trabajos [17, 18] que utilizan enfoques basados en redes neuronales para evaluar el grado de reutilización y la calidad del software a partir de diferentes métricas.

3 Plan de Trabajo

Para poder alcanzar los resultados anteriores, se plantean las siguientes actividades:

1. Analizar el estado del arte de propuestas de metamodelos de requisitos no funcionales y de integración de software.
2. Plantear las técnicas más adecuadas para metamodelar los RNF de sistemas software en las fases de requisitos y análisis, y su transformación en estilos y patrones arquitectónicos en la fase de diseño.
3. Definir metamodelos en las distintas fases para la integración de componentes software externos o previamente implementados, donde tendremos que estudiar y definir reglas que seguir a la hora de diseñar componentes software permitiendo reutilizar e integrar estos componentes.
4. Diseñar un metamodelo que permite modelar arquitecturas de sistemas distribuidos y las transformaciones de modelos conformes a este estilo arquitectónico en código (M2T).
5. Definir reglas que miden el grado de conformidad del producto obtenido en función de los RNF requeridos para un sistema y estudiar como realimentar las fases de requisitos, análisis y diseño a partir de estas medidas.
6. Desarrollar las herramientas de soporte para la creación de los modelos y su transformación automática.
7. Desplegar en un caso real los modelos y herramientas anteriores como mecanismo de validación practico de las propuestas presentadas y definir las líneas futuras de trabajo en la temática.

4 Resultados Esperados

Como resultados a este trabajo se espera, aumentar la calidad del software a través de introducir un conjunto de propiedades relacionados con la calidad (metamodelos RNF) desde la fase de captura de requisitos de manera que estas propiedades influyeran en el sistema implementado en primer lugar, automatizar las transformaciones de metamodelos de estilos y patrones arquitectónicos en código, y la integración de componentes externos permitiendo agilizar el proceso de desarrollo en segundo lugar, y verificar y validar la implementación (obtenida) respecto a los requisitos no funcionales que el sistema debe satisfacer en tercer lugar.

Para ilustrar los resultados esperados, vamos a plantear un ejemplo de aplicación de esta propuesta mediante el modelado de un sistema que requiere alto rendimiento y que el tiempo de respuesta ante un evento no supere los tres segundos. Basándonos en el metamodelo de RNF en la fase de requisitos el equipo de desarrollo puede especificar textualmente este requerimiento de alto rendimiento, que se traduce en la fase de análisis en un modelo, donde los analistas tienen que rellenar los campos del estereotipo asociado, a qué afecta al tiempo de respuesta del sistema, primero en función del uso de un recurso (CPU, datos, el ancho de banda de red, memoria, etc) donde el analista elige una opción, el CPU por ejemplo, y a continuación elegirá el número de nodos en los cuales estarán distribuidos los componentes del sistema.

Esta decisión puede ser apoyada en el caso de que el sistema reutilice componentes previamente implementados que han sido identificados en el modelo de integración de software, a partir de las métricas de ejecución que guardan estos componentes, y en función del tiempo de bloqueo para usar un recurso debido a la existencia de competencia por este recurso, que el recurso no está disponible o debido a que dependemos de otra computación que aún no ha terminado donde de nuevo el analista elige una de estas opciones y selecciona el recurso en cuestión. Esta información será traducida en la fase de diseño en un modelo acorde al estilo arquitectónico de sistemas distribuidos, reflejando decisiones tomadas en la fase anterior y haciendo que los componentes del sistema estén distribuidos en los distintos nodos lo que reduce la sobrecarga computacional y que el recurso señalado que provoca bloqueo en el sistema esté duplicado (Figura 2).

A partir de esta arquitectura y una vez seleccionados e integrados los módulos que representan los requisitos funcionales (reutilizados a partir de un repositorio o que serán implementados por los programadores) se procede a generar automáticamente el código de este sistema. Finalmente se lanzan las pruebas de rendimiento y se analizan los resultados para verificar y validar el requisito no funcional que el sistema debe satisfacer volviendo a modificar las fases anteriores y se guardan los módulos implementados (con sus respectivas métricas de ejecución) de manera que permite su reutilización.

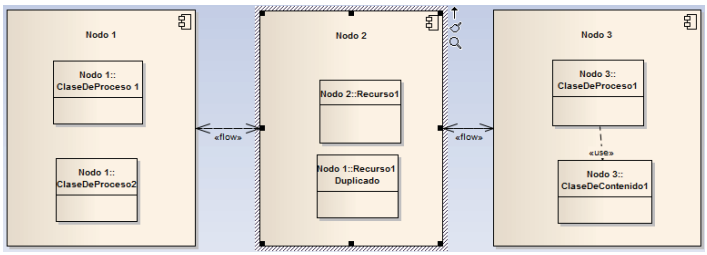


Fig. 2. Arquitectura de un sistema distribuido en la fase de diseño

References

1. Escalona, M.J., Koch, N.: Requirements engineering for web applications-a comparative study. J. Web Eng. **2**(3) (2004) 193–212
2. Batory, D.: Product-line architectures. In: Smalltalk and Java Conference. (1998)
3. Bosch, J.: Design and use of software architectures: adopting and evolving a product-line approach. Pearson Education (2000)
4. Chung, L., do Prado Leite, J.C.S.: On non-functional requirements in software engineering. In: Conceptual modeling: Foundations and applications. Springer (2009) 363–379
5. Niemelä, E., Immonen, A.: Capturing quality requirements of product family architecture. Information and Software Technology **49**(11) (2007) 1107–1120
6. Grünbacher, P., Egyed, A., Medvidovic, N.: Reconciling software requirements and architectures with intermediate models. Software and Systems Modeling **3**(3) (2004) 235–253
7. Ameller, D., Franch, X., Cabot, J.: Dealing with non-functional requirements in model-driven development. In: Requirements Engineering Conference (RE), 2010 18th IEEE International, IEEE (2010) 189–198
8. Melia, S., Cachero, C., Gomez, J.: Using mda in web software architectures. In: Proceedings of the 2 nd International Workshop on Generative Techniques in the Context of MDA., Anaheim, California, USA. (2003)
9. Gonzalez-Huerta, J., Blanes, D., Insfran, E., Abrahão, S.: Towards an architecture for ensuring product quality in model-driven software development. In: Proceedings of the 11th International Conference on Product Focused Software, ACM (2010) 28–31
10. Μάριος, M., Tsadimas, A., Τσαδήμας, A., Nikolaidou, M., Νικολαίδη, M.: Extending sysml to explore non-functional requirements: The case of information system design. (2013)
11. Kassab, M., Ormandjieva, O., Daneva, M.: A metamodel for tracing non-functional requirements. In: Computer Science and Information Engineering, 2009 WRI World Congress on. Volume 7., IEEE (2009) 687–694
12. Fernando Molina Molina; director, J.A.T.A.: Una propuesta para el tratamiento de requisitos y modelos de calidad en model driven engineering. (2010) http://hdl.handle.net/10201/34409, Murcia : Universidad de Murcia, Departamento de Informática y Sistemas (2010)
13. Prieto-Diaz, R., Freeman, P.: Classifying software for reusability. Software, IEEE **4**(1) (1987) 6–16
14. Meyer, B.: Reusability: The case for object-oriented design. IEEE software **4**(2) (1987) 50–64
15. Gill, N.S.: Importance of software component characterization for better software reusability. ACM SIGSOFT Software Engineering Notes **31**(1) (2006) 1–3
16. Yacoub, S., Ammar, H., Mili, A.: Characterizing a software component, Citeseer
17. Kumar, A.: Measuring software reusability using svm based classifier approach. International Journal of Information Technology and Knowledge Management **5**(1) (2012) 205–209
18. Manhas, S., Vashisht, R., Sandhu, P.S., Neeru, N.: Reusability evaluation model for procedure based software systems. International Journal of Computer and Electrical Engineering **2**(6) (2010) 1107–1111

People as a Service: a mobile-centric model for providing collective sociological profiles

Jose Garcia-Alonso[1], Javier Miranda[2], Javier Berrocal[1], Juan Manuel Murillo[1], and Carlos Canal[3]

[1] Department of Information Technology and Telematic Systems Engineering, University of Extremadura, Spain
{jgaralo,jberolm,juanmamu}@unex.es
[2] Gloin, Calle de las Ocas 2, Caceres, Spain
{jmiranda}@gloin.es
[3] Department of Computer Science, University of Malaga, Spain
canal@lcc.uma.es

Abstract. Mobile devices have become increasingly popular in the everyday life of many individuals. By taking an insight into the most common uses of mobile devices we clearly appreciate that accessing internet-based services has grown greatly. This, and the fact that they are extremely personal gadgets has turned them into the main interface used by individuals to express themselves towards the outside world and to receive information from others. As a result of the highly personal use, mobile devices have been granted the potential to become unrivaled devices for building and storing the virtual profiles of their owners. Access to such profiles is of great interest in fields such as governance, health, smart cities, etc. Generating a centralized profile of a user is a task upon which a lot of interest has been put in the field of social mining. People-as-a-Service (PeaaS) is a computing model that seeks to establish the foundations upon which technologies that rely on mobile-centric computing models for social purposes should evolve.

Keywords: Sociological profiles, Mobile devices, People as a Service

1 The Four Pillars of PeaaS

PeaaS aims to exploit the potential of mobile devices for storing sociological profiles of their owners, and offering them as a service. This is an alternative to traditional social computing models where users explicitly generate information that is finally centralized in servers that stand beyond their control. The model has been conceived upon four pillars describing the basic considerations which must be complied in order to guarantee that it is correctly put into practice.

Mobile devices as interfaces to people. The PeaaS model understands mobile devices as interfaces to people, which connect them to the outside world and through which they express their thoughts and concerns. This makes mobile devices a centerpiece of the PeaaS computing model.

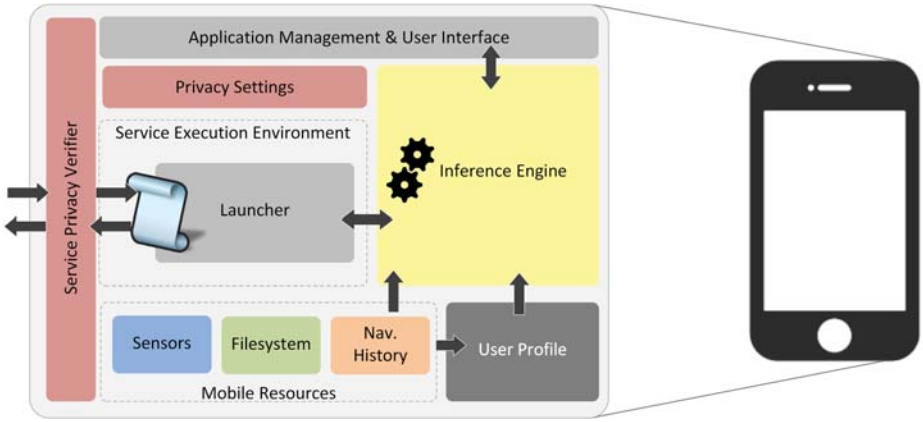


Fig. 1. High level reference architecture for PeaaS.

Sociological virtual profiles. Most mobile devices are equipped with a pack of sensors capable of collecting contextual information about their surroundings, and ergo also their owner's surroundings. The PeaaS model allows a sociological virtual profile of device owners to be built and kept in their device. Such profile is elaborated from the contextual information captured by the device's sensors and from the owner's interaction with the device.

Sociological profiles as a service. Generating profiles of the device owners is only useful to external entities if interactions with these profiles are enabled. PeaaS determines that profiles have to be provided as a service to those who may be interested in accessing this information.

User privacy. The PeaaS model guarantees that an individual's personal profile is always kept exclusively in his or her device. Furthermore, PeaaS makes all accesses to the information found within the devices completely visible and controllable by their owners.

These four pillars are combined in a mobile-centric computing model to overcome the limitations of current social computing models. Serving individuals' virtual sociological profiles through mobile devices goes one step beyond other models that focus only on providing the devices' contextualized data as a service. A reference architecture for PeaaS is shown in Figure 1 complying with the pillars described.

This paper was originally published in the IEEE Software journal and can be accessed in `http://dx.doi.org/10.1109/MS.2013.140`.

Acknowledgments

This work has been partially funded by the Spanish Government under Projects TIN2011-24278, TIN2012-34945 and TIN2012-35669. It has also been funded by the Government of Extremadura and FEDER funds.

Change-Impact driven Agile Architecting*

Jessica Díaz, Jennifer Pérez, Juan Garbajosa, Agustín Yagüe
Technical University of Madrid (UPM) - Universidad Politécnica de Madrid
CITSEM, Systems & Software Technology Group (SYST), ETSISI, Madrid, Spain
{jdiaz,jenifer.perez,jgs}@eui.upm.es, agustin.yague@upm.es

Abstract. This summary presents a solution based on the use of change-impact knowledge as the main driver for agile architecting. The solution consists of a Change Impact Analysis technique and a set of models to assist agile architects in the change (decision-making) process by retrieving the change-impact architectural knowledge resulting from adding or changing features iteration after iteration.

Keywords: Agile software development, agile architecting, change-impact analysis

1. Summary

Software architecture (SA) is a key factor to scale up Agile Software Development (ASD) in large software-intensive systems. Aligning fruitfully SA and ASD requires leveraging the inherent qualities of architectures (e.g. abstraction, communication, analysis) while complying with agile principles (e.g. open to change). This alignment can be achieved as long as practitioners are able to count on mechanisms for enabling: (i) Incremental design of features. (ii) Accommodation of new features or customizations on existing features. We refer to both of them as *agile architecting*.

Mechanisms for enabling agile architecting should assist and guide agile architects, specifically in (i) the decision-making process of implementing changes in each agile iteration, and (ii) the maintenance of the architecture integrity, i.e. the preservation of earlier architectural design decisions iteration after iteration. Regarding the former, the knowledge about the effects of a change upon the architecture provides architects with information that can be advantageously deployed to reason about how and where to implement that change. It also allows architects to make better evolution decisions based on risks, cost or viability of the change. Regarding the latter, the continuous process of architecting should never result in the software degradation as a consequence of intentionally or accidentally violation of earlier design decisions or constraints. In this sense, agile architects need knowledge about dependencies between design decisions, constraints, tradeoffs, etc., which can assist them in countering or even avoiding several well-known negative effects of software evolution—e.g. architectural erosion.

*This communication is a summary of the following published article:Díaz, J., Pérez, J., Garbajosa, Yagüe A.: Change-Impact driven Agile Architecting. Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS '13) doi: http://dx.doi.org/10.1109/HICSS.2013.127 IEEE Computer Society ISBN 978-0-7695-4892-0 (2013) pp. 4780-4789.

The presented solution[1] uses *change-impact* architectural knowledge as the main driver for agile architecting. This solution provides agile architects with knowledge to (i) assist and guide them in the *change (decision-making) process*, and (ii) favor the preservation of the architecture integrity during the iterative architecting process. This knowledge results from analyzing the impact that changes—feature increment and/or evolution—introduce into the architecture, iteration after iteration in an agile process. The solution consists of a *Change Impact Analysis* (CIA) technique and modeling artifacts for: (i) documenting architectural knowledge—the design decisions and rationale driving the iterative architecture solution—, and (ii) tracing architecturally significant features with their realization in the architecture. These models are traversed using the proposed CIA technique to retrieve the architectural design decisions and architectural components and connections that are impacted as a consequence of changing features. Therefore, these models promote communication and support (semi-)automatically reasoning over the space of architectural knowledge. This solution was built on previous work to design iteratively and incrementally software architecture (see FPLA[2]) and deployed in Scrum (see Fig. 1).

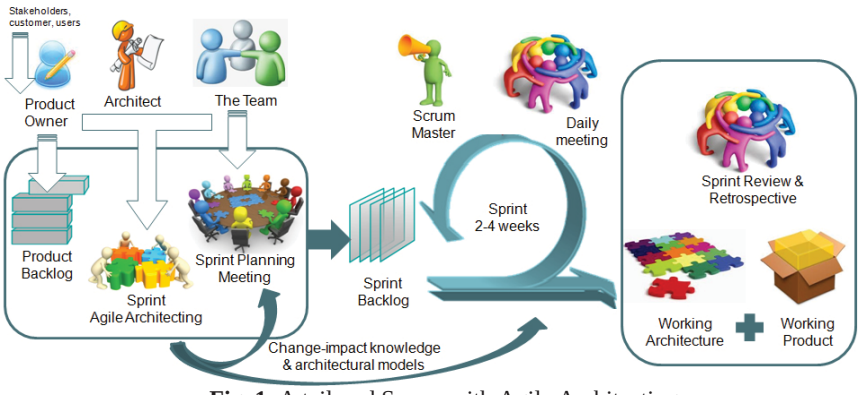


Fig. 1. A tailored Scrum with Agile Architecting

The novelty of this contribution was to prove how the output from a CIA technique can be effectively used to assist and guide agile software architecting. To empirically validate our approach we conducted a case study about smart grids in a software factory, combining both academic and industry efforts. The results prove that (i) the CIA algorithm is effective in locating the impact resulting from a change, and (ii) this change-impact helps architects to take better decisions, especially when architectural knowledge may be lost or vaporized as a result of a staff turnover. These promising results did not interfere with other agile practices and did not incur a big upfront design, making the agile construction and evolution of architecture possible. One of the main conclusions of this research is that agile architecting is feasible so that ASD can be scaled up to large and complex software-intensive systems.

[1] Sponsored by the Spanish fund (INNOSEP TIN2009-13849, IMPONET TSI-020400-2010-103, i-SSF IPT-430000-2010-038, and NEMO&CODED ITEA2 08022 IDI-20110864).
[2] The modeling framework FPLA is available on https://syst.eui.upm.es/FPLA/home

Self-Adaptation of Mobile Systems with Dynamic Software Product Lines*

Nadia Gamez, Lidia Fuentes, José María Troya

Universidad de Málaga, Dpto de Lenguajes y Ciencias de la Computación
{nadia, lff, troya}@lcc.uma.es

Abstract. Mobile ecosystems are generally long-lived and sometimes operate in inaccessible areas, so they require support for self-adaptation to the continuous context changes. Dynamic Software Product Lines (DSPLs) produce software products capable of adapting to requirements that change at runtime. We propose a DSPL-based self-adaptation process especially designed for mobile system requirements, such as context and device heterogeneity. We follow a lightweight model@run.time approach, by using variability models to generate, at runtime, the products adapted to the context changes, but without loading the models in the mobile devices. In addition, our approach takes into account the quality-of-service as part of the decision making process.

Keywords: Self-Adaptation, Dynamic Software Product Lines.

1 Overview of the Approach

One of the most appealing areas for the application of DSPL is the next generation of mobile systems. However, the majority of the contributions to the field focus on how the runtime variability is modelled and how the decisions and the reconfiguration are carried out. But none of them consider the specific problems of mobile computing self-adaptation with DSPL, as we propose in this work.

The **context definition** is central to any self-adaptation proposal. In mobile computing, the context must consider the requirements of all the actors involved in a mobile ecosystem. Context changes can trigger a system reconfiguration. In DSPL, this is defined in terms of replacing the current product of the product line with a new product adapted to the context. Nevertheless, not all the possible product configurations meet a desired quality-of-service (QoS). Therefore, the decision mechanism that chooses between all possible configurations must reach a **compromise between reconfiguration goals and the QoS**. Moreover, the reconfiguration process for mobile computing must reach an **arrangement between the use of models at runtime and an efficient reconfiguration code** and the **self-adaptation** of mobile systems must consider both **local and global reconfiguration**

* A full version of this work has been published in IEEE Software journal, 0740-7459 and partially supported by projects TIN2012-34840 funded by Spanish Gov, FEDER; P09-TIC-05231 funded by Andalusian Goverment; and FP7-317731 funded by EU, and Universidad de Málaga, Campus Excelencia Internacional Andalucía Tech.

constraints for heterogeneous devices. Finally, to achieve an effective use of DSPL by industry-specific mobile providers, we must **avoid the use of non-standard languages or formalisms,** unfamiliar to the mobile application developers.

Figure 1 shows the main steps of our DSPL-based self-adaptation process. We distinguish two main cycles: ***Context Awareness*** and ***Dynamic Reconfiguration***. Context awareness consists of observing the context changes; analysing the new context; and deciding how the system has to react to this change. If this decision implies self-adaptation, it triggers the Dynamic Reconfiguration, for selecting the new running product adapted to the context and for executing the reconfiguration.

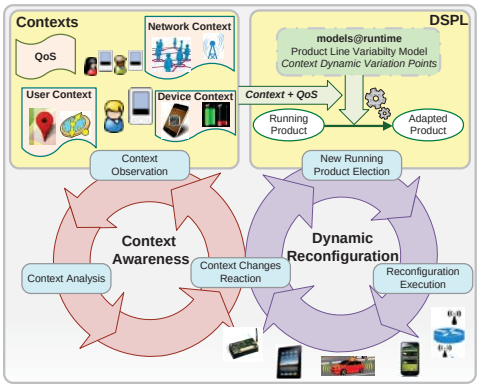


Figure 1. DSPL-based Self-adaptation mechanism for mobile ecosystems.

Context Definition: The context of mobile ecosystems at any given moment is determined by the specific contexts of all the actors that compose them. For instance, the system connectivity can be affected by the user's location (e.g., rural places), by the devices energy (e.g., low battery) or by the number of devices composing the system (e.g., thousands of devices trying to communicate). Therefore, the ***Context Observation*** needs to monitor the different contexts: *User*, *Device* and *Network*.

DSPL-Driven Reconfiguration: The first step to reconfigure the mobile systems is to choose in the product line the ***Adapted Product*** that has to replace the ***Running Product***. For this purpose, we follow a **lightweight *model@run.time*** approach, by using a *variability model* to generate, at runtime, the correct adapted configurations, but without loading the models in the devices. We use a variability model to specify the variation points that can be dynamically modified and the different context types. Avoiding the use of non-standard languages or formalisms, we use the Common Variability Language (CVL) proposed as standard by the OMG. CVL allows modeling the variability separately from the base model, but both the variability and the base models appear as connected and can be managed using the same tool. Using CVL, we generate the initial configuration and the successive adapted configurations automatically. However, it is possible that several configurations meet the new context requirements, then we also consider the QoS as part of the decision making.

Once we have obtained the adapted product, we need to perform the modifications specified at model level in the implementation level (***Reconfiguration Execution***). The main advantage of using models at runtime is that adaptations can be carried out at the model level rather than at the system level.

Automated Analysis of Diverse Variability Models with Tool Support

Fabricia Roos-Frantz, José A. Galindo, David Benavides,
Antonio Ruiz Cortés, and Jesús García-Galán

Department of Computer Languages and Systems
University of Seville
Avda. Reina Mercedes s/n, 41012, Seville, Spain
{fabriciaroos,jagalindo,benavides,aruiz,jegalan}@us.es
http://www.isa.us.es

Abstract. Over the past twenty years, there have been many contributions in the area of feature model automated analysis of variability models. We propose that the knowledge obtained during recent years to automatically analyse diverse variability models. In this paper we present FaMa OVM and FaMa DEB, which are prototypical implementations for the automated analysis of two distinct variability models, namely Orthogonal Variability Models and Debian Variablity Models, respectively. In order to minimise efforts and benefit from the feature model know–how, we use FaMa Framework which allows the development of analysis tools for diverse variability modelling languages. This framework provides a well tested system that guides the tool development. Due to the structure provided by the framework, FaMa OVM and FaMa DEB tools are easy to extend and integrate with other tools. We report on the main points of both tools, such as the analysis operations provided and the logical solvers used for the analysis.

Keywords: Variability models, Intensive variability systems, Automated analysis

1 Introduction

Variability modelling is an important task in software product line engineering (SPLE). In the literature, there are many kinds of models used for modelling variabilities [1]. The automated analysis of these models is a thriving research area. Over the past twenty years, there have been numerous contributions in this area including both, research papers and tools[2]. The automated analysis of variability models is defined as the computer-aided extraction of information from such models [2]. Some examples of analysis on models are: checking whether a configuration is valid, checking whether a model is void, etc. Although there are other variability modelling techniques, the majority of the research on analysis has focused on feature models. In this paper, we show an example of

how to implement the automated analysis of other variability models, particularly Orthogonal Variability Model (OVM) [3] and Debian Variability Model. For implementing these analysis we used the FaMa framework (FaMa FW) [4], which is an open source Java framework that simplifies the development of tools for the analysis of variability models. It is worth highlighting that in this paper we address OVMs and Debian variability models, however these are only two possibilities, amongst others. In this paper we show how we extended automated analysis beyond feature models.

This paper is structured as follows, in Section 2 we present our tools FaMaOVM and FaMaDEB and the common parts they share. The concretes parts of each tool are described in subsection 2.1 for FaMaOVM and in subsection 2.2 for FaMaDEB. Finally in Section 3 we present our conclusions and future work.

2 The tools

FaMa OVM and FaMa DEB are tools for the automated analysis of OVMs and Debian Variability Models, respectively. Both tools are extensions of the FaMa FW. This framework provides a number of extension points to plug in new components called metamodels, readers/writers and reasoners. FaMa FW provides a set of analysis operations to observe the properties of a model without modifying it. Each operation takes a model as input and provides a response as result. More details about analysis operations can be found in [2]. FaMa OVM and FaMa DEB operations:

- *Void model*: checks whether a model is void or not, i.e. if it represents at least one valid product. A model may become void due to the wrong usage of constraints.
- *#Products*: returns the total number of valid products represented by the model received as input.
- *Valid Product*: takes a model and a product (set of variability elements) as input and returns a value that determines whether the input product belongs to the set of products represented by the model or not.
- *All Products*: takes as input a model and returns all the valid products represented by this model.
- *Valid Partial Configuration*: takes a model and a partial configuration as input and returns a value informing whether the partial configuration is valid or not, i.e. a partial configuration is valid if it does not include any contradiction.
- *Filter*: takes as input a model and a configuration (potentially partial) and returns the set of valid products including the input configuration that can be derived from this model.
- *Dead Node*: returns a set of dead nodes (if any), i.e. those that cannot appear in any of the valid products represented by the model. Dead nodes are caused by the wrong usage of constraint dependencies.
- *Commonality*: takes a model and a configuration as inputs and returns the percentage of valid products including the input configuration.

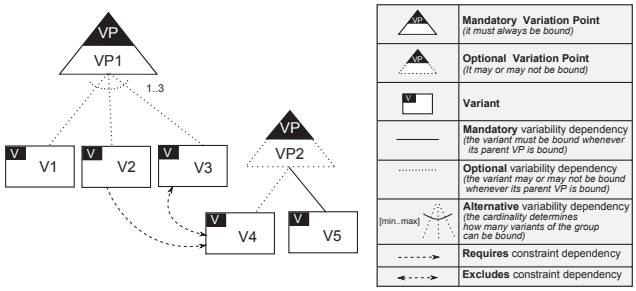


Fig. 1. OVM notation

2.1 The FaMa OVM tool

OVM is a modelling language for defining the variability of a software product line separately without change the base models (e.g. requirement model, design model). The base models realise the variability defined by the variability elements in the OVM model. In OVM the first-classes are: *variation points (VP)* and *variants*. A variation point documents the aspects that can vary in the product line, which must be chosen by the customer or engineer of the software product line. A variant is related to a variation point and documents how this variation point can vary. Figure 1 shows an example of an OVM diagram. We have successfully used this tool to analyse OVM models using attributes in an industrial case study in the automotive domain [5]. The main ideas behind the mapping of OVM to constraint programming were presented in [5].

The OVM metamodel. → In the context of automated analysis, an OVM is interpreted as a set of possible products that can be derived from the product line. Thus, the OVM metamodel describes the different variability elements and the rules that constraint the combination of these elements in a product line.

2.2 The FaMa DEB tool

We found that Debian–based Linux distributions use a language to describe dependencies in software packages repositories. Later, we realised that this language can be interpreted as a variability language similar to those of feature models, OVMs and other variability description languages. If we consider a software package as a feature of a distribution, a Debian–based Linux distribution could be interpreted as a product line and the individual installations could be therefore considered the products of the product line. In this context, a product is a set of installed packages, i.e. a Debian installation. Similarly, we assume that a set of repositories define an SPL of Debian-based distributions.

The DEB Metamodel. → The FaMa DEB metamodel is needed to represent and store the information present into Debian repositories. It is composed by packages (varibility elements) and relationships. Note that not all information stored in the Debian repository describes the variability of the linux distribution

(i.e. the homepage of the software maintainer). In [6] you can find a detailed description of those relations and the required mapping.

3 Conclusion and Future Work

We provided a prototypical implementation for the automated analysis of OVMs and Debian variability models. The development of these tools were based on a framework for the analysis of feature models, which is a research area with more know–how. We realised that the framework simplified the development of our tools, we did not have to start from scratch and we reused most of the analysis operations applied to feature models. Both tools offer a simple public interface, by which the user just need to entry with the model (sometimes with some more data, it depends on the operation) and to select the operation to be answered by the tool. It is worth highlighting that, on the one hand, we used small–scale models to test FaMa OVM and the analysis results were obtained in an acceptable time for all analysis operations performed with both sat4j and choco solvers. On the other hand, with FaMaDEB we were able to deal with realistic large-scale variability models.

Acknowledgment

The prototypes of FaMa OVM and FaMa DEB are available at `http://www.lsi.us.es/~dbc/material/jisbd14/`. This work has been partially supported by the European Commission (FEDER) and Spanish Government under CICYT project TAPAS (TIN2012-32273) and the Andalusian Government projects THEOS (TIC-5906) and COPAS (P12-TIC-1867).

References

1. L. Chen, M. A. Babar, and N. Ali, "Variability management in software product lines: a systematic review," in *SPLC*, 2009, pp. 81–90.
2. D. Benavides, S. Segura, and A. Ruiz-Corts, "Automated analysis of feature models 20 years later: a literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010. [Online]. Available: http://dx.doi.org/10.1016/j.is.2010.01.001
3. K. Pohl, G. Böckle, and F. J. van der Linden, *Software Product Line Engineering: Fundations, Principles and Techniques.* Berlin, DE: Springer–Verlag, 2005.
4. P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A.Jimenez, "Fama framework," in *12th Software Product Lines Conference (SPLC)*, Sep 2008, p. 359.
5. F. Roos-Frantz, D. Benavides, A. Ruiz-Corts, A. Heuer, and K. Lauenroth, "Quality-aware analysis in product line engineering with the orthogonal variability model," *Software Quality Journal*, pp. 1–47. [Online]. Available: http://dx.doi.org/10.1007/s11219-011-9156-5
6. J. A. Galindo, D. Benavides, and S. Segura., "Debian packages repositories as software product line models. towards automated analysis," in *Proceeding of the First International Workshop on Automated Configuration and Tailoring of Applications (ACOTA)*, 2010.

WindRose: A Cloud Based IDE for the Automated Analysis of Feature Models

José A. Galindo, David Benavides, Mauricio Alférez, Mathieu Acher and Benoit Baudry

Department of Computer Languages and Systems
University of Seville
Avda. Reina Mercedes s/n, 41012, Seville, Spain
fabriciaroos,jagalindo,benavides,aruiz@us.es
http://www.isa.us.es

Abstract. Feature modeling is the "de facto" standard to describe the common and variant parts of software product lines. Different tools, approaches and operations for the automated analysis of feature models have been proposed in the last 20 years. However, the installation and usage of those tools use to be time consuming. In this paper we present the WindRose IDE, a cloud based IDE that allows the storage, edition and analysis of feature models while being executed in the cloud. WindRose integrates different feature model analysis operations such as Valid or Number of Products. This reasoning capabilities rely in different well stablished tools like FaMa or FaMiliar.

Keywords: Variability Modeling, Tool integration, Automated Analysis

1 Introduction

To extract useful information from feature models, researchers and practitioners proposed the use of computer-aided techniques. This is commonly called Automated Analysis of feature models. In the last 20 years of feature modeling, practitioners released different analysis tools and more than 30 operations were proposed in the literature such as "NumberOfProducts" or "ValidModel".

Currently there is a diversity of tools, operations and textual formats. However, the tediousness of configuring each tool hinders the collaboration and reuse of them. In this paper, we present the WindRose tool. WindRose is a cloud-based ide that enables the analysis operations execution in the cloud. This easies off the requirement of installing and configuring each tool and enables the collaboration between practitioners when designing a feature model.

The main features of WindRose are:

- Allows the simultaneous editing of feature models in textual language.
- Enables the reasoning of feature models using well-tested tools such as FaMa or FaMiLiAr.

– Offers two repositories, the first public, where all users can share their models, the second private, where researchers can build their models without prying eyes.
– Is open source, this is, practitioners can install it in their own server for private use.

2 WindRose cloud-based IDE

WindRose is a cloud-based IDE that integrates different automated reasoning tools. It has been developed on top of the CODIAD IDE (`http://codiad.com/`) which has been developed by the MIT and provides useful collaboration tools and a plugin system out of the box.

2.1 Available plugins

WindRose offers out of the box integration with different analysis tools. In WindRose, each plugin is associate with one textual format. This is used to filter by the file extension the available analysis operations. Some plugin implementations are:

– *FaMaTS-SAT*. FaMa [2] is a framework for building variability analysis tools. Different tools have been build on top of it's architecture such us FaMaDEB, FaMaOVM and Betty. It also offers different ways of consume its services: i) by OsGi; i) by Web services (REST and SOAP); iii) by terminal and iv) by importing the libraries in a java program. Concretely we configured this tool to use the SAT4j [3] based implementation of the analysis operations for traditional feature models. This tool is associated with the files having the .fama extension.
– *FaMaTS-CSP for Attributed FM*. FaMa enables the analysis of attributed feature models. Internally this plugin uses the Choco2 solver `http://www.emn.fr/z-info/choco-solver/` as back-end for executing the analysis operations. This plugin is associated with the extension .afm and .vm
– *FaMiLiAr*. FaMiLiar [1] is a framework for managing families of products while enabling operations such as merging different models. This plugin is associated with the extension .tvl.
– *VANE*. This is a proposal developed by INRIA Rennes in France. It allows to obtain the pair-wise coverages that maximizes a custom attribute value [4].
– *TESALIA*. This is a proposal for the testing of feature models containing quality information. This tool is also used internally as part of the ATAACK cloud [5] and help with the testing of Android apps. This plugin is associated with the afm file format.

2.2 Analysis operations

WindRose offers out of the box different feature model operations. Note that these operation can be extending by implementing new plugins. Currently the supported operations are:

- *Valid Model.* This operations checks the semantic correctness of a feature model. This is, if it represent at least one product. This operation is implemented by the FaMaTS-SAT, FaMaTS-CSP for Attributed FM and FaMiLiAr plugins.
- *Valid Configuration.* This operation takes a model and a partial configuration and check if the configuration is correct or not. This operation is implemented by the FaMaTS-SAT and FaMaTS-CSP for Attributed FM plugins.
- *Valid Product.* This operation takes a product (a.k.a non-partial configuration) and checks its validity on top of the selected model. This operation is implemented by the FaMaTS-SAT and FaMaTS-CSP for Attributed FM plugins.
- *All products.* This operation print out the list of valid products from a feature model. FaMaTS-SAT, FaMaTS-CSP for Attributed FM and FaMiLiAr plugins.
- *Dead Features.* This operation detects those features that cannot be selected because of the topology of the model. This operation is implemented by the FaMaTS-SAT and FaMaTS-CSP for Attributed FM plugins.
- *Core Features.* This operation returns the features present in all products. This operation is implemented by the FaMaTS-SAT and FaMaTS-CSP for Attributed FM plugins.
- *Pair-Wise.* This operation returns a random pairwise coverage.This operation is implemented by the VANE plugin.
- *Pair-Wise with attributes.* This operation returns the pair-wise coverage that maximize a concrete attribute value. This operation returns a random pairwise coverage.This operation is implemented by the VANE plugin.
- *Prioritization.* This operation returns the list of products but ordered by a function between attributes. This operation returns a random pairwise coverage. This operation is implemented by the TESALIA plugin.
- *Packaging.* This operation apply a knapsack over products containing value and cost definition. This operation is implemented by the TESALIA plugin.
- *Pruning.* This operation returns the set of products that meet a custom constraint over attributes. This operation is implemented by the TESALIA plugin.

2.3 Repositories

WindRose offers two repositories. In the first hand, a public repository that does not require registration and allows users to download any model stored and execute analysis operations depending on the size of the models. For example, the operation that retrieves all valid configurations from a model is restricted to small-size models in the public repository because of the large computing capabilities required to execute the operation. However, models in the public repository can not be overwritten. In the second hand, registered users can work privately with all-size models. Also these models can be moved to the public repository so they can be publicly released. Figure1 shows some screenshots of WindRose public repository.

(a) WindRose home page

(b) WindRose public repository FaMaTS-SAT execution

Fig. 1: Screenshots of WindRose in action

Acknowledgements

While is possible to download and install WindRose in private servers inside an organization we provide an instance of it for the review process. To check it out go to `http://windrose.github.io/` and click in the link to the cloud ide. This work has been partially supported by the European Commission (FEDER), the Spanish Government under project TAPAS (TIN2012-32273) project and the Andalusian Government under projects THEOS (TIC-5906) and COPAS (P12-TIC-1867).

References

1. Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming (SCP) Special issue on programming languages*, page 22, 2013.
2. David Benavides, Pablo Trinidad, Antonio Ruiz Cortés, and Sergio Segura. *FaMa*, chapter FaMa, pages 163–171. Springer Berlin Heidelberg, 2013.
3. Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2. *JSAT*, 7(2-3):59–6, 2010.
4. José A. Galindo, Mauricio Alférez, Mathieu Acher, and Benoit Baudry. A variability-based testing approach for synthesizing video sequences. In *ISSTA*, To appear in 2014.
5. Hamilton Turner, Jules White, Jeff Reed, José A. Galindo, Adam Porter, Madhav Marathe, Anil Vullikanti, and Aniruddha Gokhale. *Building a Cloud-based Mobile Application Testbed*. IGI Global, 701 E. Chocolate Avenue Hershey PA 17033-1240, USA, 2012.