novática
Revista de la Asociación de Técnicos de Informática

Novática, revista fundada en 1975 y decana de la prensa informática española, es el órgano oficial de expresión y formación continua de **ATI** (Asociación de Técnicos de Informática), organización que edita también la revista **REICIS** (Revista Española de Innovación, Calidad e Ingeniería del Software).

<http://www.ati.es/novatica/>
<http://www.ati.es/reicis/>

ATI es miembro fundador de **CEPIS** (*Council of European Professional Informatics Societies*) y es representante de España en **IFIP** (*International Federation for Information Processing*); tiene un acuerdo de colaboración con **ACM** (*Association for Computing Machinery*), así como acuerdos de vinculación o colaboración con **AdaSpain**, **AI2**, **ASTIC**, **RITSI** e **Hispalinux**, junto a la que participa en **ProInnova**.

Coordinación Editorial, Redacción Central y Redacción ATI Madrid
Padilla 66, 3º, dcha., 28006 Madrid
Tlfn.914029391; fax.913093685 <novatica@ati.es>
Composición, Edición y Redacción ATI Valencia
Av. del Reino de Valencia 23, 46005 Valencia
Tlfn. 963740173 <novatica_prod@ati.es>
Administración y Redacción ATI Cataluña
Via Laietana 46, ppal. 1ª, 08003 Barcelona
Tlfn.934125235; fax 934127713 <secregen@ati.es>
Redacción ATI Aragón
Lagasca 9, 3-B, 50006 Zaragoza.
Tlfn./fax 976235181 <secreara@ati.es>
Redacción ATI Andalucía <secreand@ati.es>
Redacción ATI Galicia<secregal@ati.es>
Suscripción y Ventas <novatica.subscripciones@atinet.es>
Publicidad Padilla 66, 3º, dcha, 28006 Madrid.
Tlnf.914029391; fax.913093685 <novatica@ati.es>
Imprenta: Derra S.A., Juan de Austria 66, 08005 Barcelona.
Depósito legal: B 15.154-1975 -- ISSN: 0211-2124; CODEN NOVAEC
Portada: Escalera de color - Concha Arias Pérez / © ATI
Diseño: Fernando Agresta / © ATI 2003

Nº 221, enero-febrero 2013, año XXXIX

sumario

Tema del próximo número: "Lenguajes de programación"

Presentación. Enfoques actuales para el desarrollo de software modular

Mercedes Amor Pinilla, Lidia Fuentes Fernández, Mónica Pinto Alarcón
Departamento de Lenguajes y Ciencias de la Computación, ETSI Informática, Universidad de Málaga

<{pinilla,lff,pinto}@lcc.uma.es>

Una preocupación recurrente en el desarrollo de sistemas software ha sido y sigue siendo cómo conseguir una buena modularización o desarrollo modular de nuestro sistema, con el objetivo de desarrollar sistemas más mantenibles y adaptables.

La modularización tiene diferentes definiciones atendiendo a la disciplina en la que sea considerada. De hecho, dentro de una misma disciplina como es la Ingeniería del Software, tampoco hay consenso sobre una única definición. Así, en una Retrospectiva sobre Modularidad (<http://aosd.net/2011/retrospective_on_modularity.html>) que tuvo lugar en la conferencia AOSD (*Aspect-Oriented Software Development*) 2011: *Perspectives on Modularity*, se pidió a un cierto número de investigadores y profesionales del desarrollo de software sus propias definiciones del término "modularidad", intentando que esta definición fuera "lo más amplia pero precisa posible". Las definiciones dadas fueron muy diversas, pero todas coincidían en que la modularidad es una propiedad que permite descomponer, organizar y estructurar los sistemas en módulos atendiendo a diferentes dimensiones, como por ejemplo, el tamaño de los módulos, su funcionalidad o las dependencias entre ellos. Entre los beneficios esperados, una adecuada modularización permite incrementar la reutilización, así como facilitar la evolución y mantenimiento del sistema.

Esta definición general de modularidad concuerda con el enunciado del principio de Separación de Conceptos (SoC, *Separation of Concerns* en inglés), uno de los fundamentos de diseño clásicos en la Ingeniería del Software. Este principio de diseño aboga por separar de forma adecuada las diferentes propiedades funcionales y extra-funcionales de un sistema en módulos independientes.

La modularidad ligada a una adecuada separación de conceptos se puede conseguir de muy diversas formas, siendo las más básicas el uso de subrutinas o la programación orientada a objetos. En la actualidad hay muchas tecnologías de desarrollo que tienen como principal objetivo conseguir una mejor modularidad de los sistemas, como son la separación en componentes (*component-based separation*), el uso de aspectos (*aspect-orientation*), el uso de líneas de productos software (*software product lines*) o el desarrollo dirigido por modelos (*model driven development*).

Con el objetivo de cubrir cómo se concibe la modularidad en diferentes ámbitos, pero siendo conscientes de que han quedado otros muchos por abordar, esta monografía ha recogido el trabajo, la experiencia y puntos de vistas de expertos en diferentes áreas acerca de la importancia de modularizar en cualquier fase del desarrollo.

Así, los dos primeros trabajos tratan la modularidad aplicando el enfoque de Desarrollo de Software Orientado a Aspectos (AOSD, en inglés *Aspect-Oriented Software Development*). El AOSD se ocupa de la identificación de propiedades transversales, que son aquellas propiedades que aparecen dispersas o entremezcladas en diferentes módulos de un sistema. El objetivo es separarlas y encapsularlas en un nuevo módulo llamado aspecto. El comportamiento de un aspecto es entremezclado de nuevo mediante un proceso de composición, denominado entretejido en la terminología de aspectos, en ciertos puntos del sistema que se pueden expresar mediante los denominados puntos de corte.

Editoras invitadas

Mercedes Amor Pinilla es Profesora Contratada Doctora en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga. Recibió su título de Ingeniera Informática en 1998 por la Universidad de Málaga, y es Doctora por dicha Universidad desde 2005. En la actualidad, sus principales líneas de investigación tienen que ver con la aplicación de las tecnologías avanzadas en el desarrollo de agentes software para entornos de Inteligencia Ambiental y el Internet de las Cosas (Desarrollo de Software Orientado a Aspectos, Desarrollo Dirigido por Modelos, Computación Autonómica, etc.). Ha participado y participa actualmente en distintos proyectos nacionales y europeos sobre AOSD (AOSD-Europe, AMPLE, INTER-TRUST, etc.). Es miembro del grupo de investigación CAOSD (*Component and Aspect Oriented Software Development Group*).

Lidia Fuentes Fernández se licenció como Ingeniera en Informática por la Universidad de Málaga en 1992 y recibió el grado de doctora Ingeniera en Informática por la misma Universidad en 1998. Ha ocupado diversos puestos como profesora del departamento de Lenguajes y Ciencias de la Computación desde 1993 siendo actualmente Catedrática de Universidad. Sus principales líneas de investigación tienen que ver con la aplicación de las tecnologías avanzadas en el desarrollo de software (por ej. Desarrollo de Software Orientado a Aspectos, el Desarrollo Dirigido por Modelos, las Líneas de Producto Software, etc.) a los sistemas distribuidos y pervasivos. Su producción científica es muy prolífica, con más de cien publicaciones en foros de prestigio internacional, incluyendo artículos en revistas de impacto como *IEEE Internet Computing*, *IEEE Transactions on Software Engineering, Information and Software Technology* y *ACM Computing Surveys* entre otras. Su trabajo es igualmente muy citado con más de mil quinientas citas. Lidera el *Analysis and Design lab* en la post-EC phase del proyecto AOSD-Europe. Ha sido miembro del comité de programa de múltiples conferencias relevantes en Ingeniería del Software como AOSD, OOPSLA, MODELS o GPCE. Además, lidera y participa activamente en diversos proyectos de investigación, resaltando aquellos europeos como AOSD-Europe, AMPLE o INTER-TRUST. Lidera actualmente el grupo de investigación CAOSD (*Component and Aspect Oriented Software Development Group*) <http://caosd.lcc.uma.es>.

Mónica Pinto Alarcón es Profesora Titular de Universidad en el Departamento de Lenguajes y Ciencias de la Computación de la Universidad de Málaga, en España. Recibió su título de Ingeniera Informática en 1998 por la Universidad de Málaga, y es Doctora por la Universidad de Málaga desde 2004. Sus principales líneas de investigación son la Ingeniería de Software Basada en Componentes, el Desarrollo de Software Orientado a Aspectos, los Lenguajes de Descripción de Arquitecturas Software, el Desarrollo Dirigido por Modelos y las Plataformas Móviles Sensibles al Contexto. En los últimos años ha organizado el taller "*Early Aspects*" en la conferencia ICSE, y ha sido miembro del comité de programa de varios talleres y conferencias en conferencias sobre AOSD y composición de software. Ha sido "*publicity chair*" en la conferencia AOSD 2011 y AOSD 2012. Ha participado y participa actualmente en distintos proyectos nacionales y europeos sobre AOSD (e.g. AOSD-Europe, AMPLE, INTER-TRUST, etc.). Es miembro del grupo de investigación CAOSD (*Component and Aspect Oriented Software Development Group*).

La limitación de muchas técnicas actuales de separación de aspectos es que no se fundamentan en una definición formal y precisa de estas propiedades transversales, lo que es clave para su correcta identificación. En ese sentido, el primer artículo, de ***José María Conejero Manzano*** *y* ***Juan Hernández Nuñez***, de la Universidad de Extremadura, define un marco de trabajo conceptual para la definición formal de estos conceptos y presenta dos aplicaciones del mismo, como son la identificación de propiedades transversales a nivel de requisitos y la minería de aspectos.

Aunque el AOSD cubre todas las fases del ciclo de vida, la separación de aspectos surgió en el nivel de implementación, en lo que se conoce como Programación Orientada a Aspectos (AOP, en inglés *Aspect Oriented Programming*). La división en módulos independientes mejora la reutilización, comprensibilidad, extensibilidad y mantenimiento del código. En este contexto, el artículo de ***Francisco Ortín Soler***, de la Universidad de Oviedo nos presenta una visión didáctica de AOP, ofreciendo una comparativa entre AOP y orientación a objetos ilustrada de forma muy amena mediante la implementación de un problema típico utilizando ambos paradigmas. Para implementar la separación de aspectos usa AspectJ, uno de los lenguajes orientados a aspectos más maduros y utilizados actualmente, tanto a nivel académico como industrial.

Otra técnica para mejorar la modularidad de un sistema es la Ingeniería Dirigida por Modelos (MDE, en inglés *Model Driven Engineering*). El MDE se centra en la separación en distintos modelos de las características relevantes de un sistema y en las transformaciones entre modelos. En este sentido, el tercer artículo, de ***Paulo F. Pires, Flávia C. Delicato, y Jesús Martín Talavera Portocarrero***, de la Universidade Federal do Rio de Janeiro en Brasil, combina el AOSD con el MDE para resolver un problema del AOSD conocido como fragilidad de los puntos de corte. Este problema se produce cuando los puntos de corte definidos en un sistema dejan de ser válidos ante una evolución del sistema. La propuesta presentada en este artículo resuelve este problema abstrayendo la separación y entretejido de aspectos y definiéndolos en función de modelos de alto nivel. En concreto, se describe un modelo de aspectos donde los puntos de corte se definen de forma abstracta a nivel de un modelo conceptual del sistema. Posteriormente, haciendo uso de un conjunto de transformaciones MDE estos modelos se componen entre sí y con el modelo base de la aplicación con el objetivo de generar un modelo conjunto a nivel de diseño.

Otro elemento clave del MDE son los lenguajes que se ocupan de especificar e implementar las transformaciones entre modelos, que deben proporcionar construcciones modulares que permitan su reutilización y extensión. En esta línea, el cuarto artículo, de ***Jesús J. García Molina y Jesús Sánchez Cuadrado***, de la Universidad de Murcia, analiza la modularidad ofrecida por los lenguajes de transformación en enfoques dirigidos por modelos utilizando como ejemplo el lenguaje RubyTL.

Otro de los artículos interesantes y prácticos de esta monografía trata acerca de los beneficios de una modularización adecuada en el ámbito del desarrollo, evolución y mantenimiento de aplicaciones web. Los autores, ***Antonia M. Reina Quintero***, ***Rafael Corchuelo Gil***, y ***Miguel Toro Bonilla***, de la Universidad de Sevilla, ofrecen una visión general de los diferentes enfoques y tecnologías web que, aplicando el principio de separación de conceptos, mejoran la modularidad de este tipo de aplicaciones. Además, contribuyen a mejorar características específicas de las aplicaciones web, como el diseño de la navegación, lo que acaba influyendo positivamente en la experiencia del usuario.

Por último, recientemente se han empezado a aplicar con éxito otras tecnologías que contribuyen de forma positiva a la modularidad de un sistema, como son las Líneas de Producto Software (SPL, en inglés *Software Product Line*), dirigidas principalmente al desarrollo de software en el ámbito empresarial. La importancia de una buena modularización en el ámbito de desarrollo de software industrial es abordada en el último artículo de esta monografía, de ***Salvador Trujillo González*** de Ikerland y ***David Benavides Cuevas*** de la Universidad de Sevilla. En este artículo se muestra como el enfoque de líneas de productos software ofrece los mecanismos necesarios para el desarrollo y configuración de sistemas de domótica de forma más efectiva.

Referencias útiles sobre "*Modularidad*"

A continuación se proporcionan algunas referencias, que complementan a las incluidas en cada uno de los artículos, y que permitirán al lector profundizar más en los distintos enfoques de desarrollo modular tratados en esta monografía.

Revistas

- **IEEE Transactions on Software Engineering.** <http://www.computer.org/portal/web/tse/> (IEEE).
- **IET Software.** <http://digital-library.theiet.org/content/journals/iet-sen> (IET).
- **Information & Software Technology.** <http://www.journals.elsevier.com/information-and-software-technology/> (ELSEVIER).
- **Journal of Systems and Software.** <http://www.journals.elsevier.com/journal-of-systems-and-software/> (ELSEVIER).
- **Software and System Modelling.** <http://www.sosym.org/ > (SPRINGER).

Congresos más relevantes

- **AOSD.** International Conference on Aspect-Oriented Software Development, <http://www.aosd.net/2013/>.
- **CBSE.** International Symposium on Component-Based Software Engineering, <http://cbse-conferences.org/>.
- **ECOOP.** European Conference on Object-Oriented Programming, <http://www.ecoop. org/>.
- **ECSA.** European Conference on Software Architecture, < http://www.lirmm.fr/ecsa13/>
- **ICSE.** International Conference on Software Engineering, <http://2013.icse-conferences. org>
- **ICWS.** IEEE International Conference on Web Services, <http://conferences. computer. org/icws/>.
- **MODELS.** International Conference on Model Driven Engineering and Systems, < http://www.modelsconference.org/>.
- **SPL.** International Software Product Line Conference, <http://www.splconf.org/>.
- **WICSA.** Working IEEE/IFIP Conference on Software Architecture, <http://www. wicsa.net/>

Asociaciones

- **AITO.** *Association Internationale pour les Technologies Objets*, <http://www.aito.org/>.
- **AOSA.** *Aspect-Oriented Software Association*, <http://www.aosd.net/aosa.php>.
- **AOSD-*Europe Post-funding Association*.** European Network of Excellence on Aspect-Oriented Software Development Post-funding Association, <http://www.aosd-europe.org>

GLOSARIO DE TÉRMINOS

Término en inglés	Traducciones al español	Definición
concern	Asuntos, intereses, propiedades, conceptos	Requisitos o características demandados a una aplicación.
tangling/tangled concern	Asunto/interés entremezclado con otros	Asuntos/intereses que se encuentran entremezclados (*tangled*) con otros asuntos/intereses en un mismo artefacto software.
scattering/scattered concern	Asunto/interés disperso/repartido entre otros	Asuntos/intereses que se encuentran repartidos (*scattered*) entre distintos artefactos software.
crosscutting concern	Asunto/interés transversales a otros	Definición 1: Asuntos/intereses cuyo diseño/implementación es transversal (crosscut) al diseño/implementación de otros asuntos/intereses. --------- Definición 2: Asuntos/intereses que se encuentran: (1) entremezclados con otros asuntos/intereses en un mismo artefacto software, y/o (2) repartidos entre distintos artefactos software.
separation of concerns	Separación de conceptos ó Separación de intereses	Principio de diseño que trata de dividir los asuntos/intereses en módulos software independientes.
aspect	Aspecto	Este es el término usado a nivel de implementación para llamar a los asuntos transversales (*crosscutting concerns)*.
joinpoint	Punto de intercepción	Puntos de un artefacto software que pueden ser interceptados por un asunto/interés (ej. una interacción entre dos componentes a nivel de diseño, la llamada o ejecución de un método a nivel de implementación, etc.).
pointcut	Punto de corte	Es un predicado (en forma por ejemplo de expresión regular) que permite identificar y seleccionar un conjunto de puntos de intercepción (joint points).
advice	Comportamiento del aspecto	Comportamiento que el aspecto incorpora a los puntos de intercepción identificados por un punto de corte. Pueden ser de distintos tipos (ej. *before*, *after*, *around*).
aspect weaver	Tejedor de aspectos	Herramienta que se encarga de tomar los módulos que implementan los distintos aspectos y generar la aplicación final.
weaving	Composición	Proceso de composición entre los asuntos/intereses/aspectos y los artefactos software (ej. arquitecturas software, código, etc.) que modelan o implementan la funcionalidad básica de la aplicación.
symmetric AOSD	AOSD simétrica	Técnica AOSD donde la separación entre intereses/aspectos se realiza de forma simétrica. Para ello se modelan/implementan los aspectos utilizando el mismo artefacto software que es utilizado para modelar los comportamientos que no son transversales.
asymmetric AOSD	AOSD asimétrica	Técnica AOSD donde la separación entre intereses/aspectos se realiza de forma asimétrica. Para ello se modelan/implementan los aspectos utilizando un artefacto software específico. Por ejemplo, un lenguaje de programación asimétrico es un lenguaje donde los objetos/componentes encapsulan el comportamiento de la aplicación que no es transversal a otros y un nuevo artefacto software llamado aspecto modela el comportamiento transversal.
feature model	Modelo de características	Modela el dominio de aplicación de una línea de productos software, identificando las características comunes a todos los productos de la familia, si son obligatorias u opcionales, y las dependencias entre ellas.
platform independent model (PIM)	Modelo independiente de la plataforma	En un enfoque MDE, modelo del sistema que describe su funcionalidad de forma independiente de la plataforma o tecnología utilizada para implementarlo.
platform specific model (PSM)	Modelo específico de la plataforma	En un enfoque MDE, modelo del sistema que describe su funcionalidad para una plataforma o tecnología específica.