Información Tecnológica
Vol. 29(6), 213-220 (2018)
http://dx.doi.org/10.4067/S0718-07642018000600213

Propuesta de Modelo de Características con Interfaz de Punto de Unión para el Modelamiento de Líneas de Productos de Software

Cristian L. Vidal-Silva[(1)], Miguel A. Bustamante[(2)]*, José M. Rubio[(3)], Luis E. Carter[(4)]
(1) Ingeniería Civil Informática, Escuela de Ingeniería, Campus Rodelillo, Universidad Viña del Mar, Agua Santa 7055, Viña del Mar-Chile. (e-mail: cristian.vidal@uvm.cl)
(2) Facultad de Economía y Negocios, Universidad de Talca, Avenida Lircay S/N, Talca-Chile. (e-mail: mabu@utalca.cl)
(3) Área Académica de Informática y Telecomunicaciones, Universidad Tecnológica de Chile INACAP, Av. Vitacura 10.151, Vitacura, Santiago – Chile. (e-mail: jrubiol@inacap.cl)
(4) Facultad de Ingeniería, Ingeniería Civil Industrial, Universidad Autónoma de Chile, 5 Norte 1670, Talca-Chile. (e-mail: luis.carter@uautonoma.cl)

* Autor a quien debe ser dirigida la correspondencia

Recibido Abr. 18, 2018; Aceptado Jun. 19, 2018; Versión final Jul. 10, 2018, Publicado Dic. 2018

Resumen

En búsqueda de una nueva metodología de desarrollo de software modular, este trabajo propone MC JPI, esto es, Modelos de Características (MC) con Interfaz de Punto de Unión (JPI del inglés Join Point Interface) de Programación Orientada a Aspectos (POA) como base de la metodología Programación Orientada a la Característica (FOP) + JPI. Así, este trabajo describe ventajas y detalles de FOP y POA JPI como paradigmas individuales y de su simbiosis para la producción de software modular. Como aplicación ejemplo, se utiliza MC JPI sobre un ejemplo clásico de FOP para visualizar componentes propios de MC tradicionales, y asociaciones o restricciones cruzadas especiales entre características. Adicionalmente, este trabajo describe las ventajas de MC JPI y las diferencias respecto a algunos trabajos de investigación anteriores para apoyar principios de modelado orientados a aspectos propios de JPI. Se concluye que JPI permite una modularización sobre modelos de características, y así este enfoque de simbiosis JPI + FOP parece prometedor.

Palabras clave: modelo de características, características, aspectos, JPI, FOP, POA.

A Proposal of Feature Model with Join Point Interface for the Modeling of Software Product Lines

Abstract

Looking for a new methodology of modular software development, this paper proposes JPI FM, that is, Feature Models (FM) with Join Point Interfaces (JPI) of Aspect-Oriented Programming (AOP) as the basis of the Feature-Oriented Programming (FOP) + JPI methodology. Therefore, this paper describes the advantages and details of FOP and AOP JPI as individual paradigms and their symbiosis to produce modular software. As an application example, MC JPI is used over a classic FOP example to visualize traditional FM components, and special associations or constraints between features. Additionally, this paper describes the advantages of JPI FM and the differences with respect to some previous research works to support aspect-oriented modeling principles of JPI. This paper concludes that JPI allows modularization on feature models, and thus this approach seems promising.

Keywords: features model, features, aspects, JPI, FOP, AOP.

INTRODUCCIÓN

Un Modelo de Características (MC) representa las características y relaciones entre características de una Línea de Productos de Software LPS (Benavides et al., 2010; Vidal et al., 2015; Vidal et al., 2015b; Vidal et al., 2015c). Un MC ilustra la similitud y variabilidad de los productos software de una LPS. De acuerdo con (Benavides et al., 2010; Apel et al., 2013), un producto válido de una LPS representa una selección consistente de un conjunto de características, donde cada característica representa un incremento de funcionalidad. Cada instancia de MC establece posibles selecciones de características para la definición válida de productos. El proceso de selección de características se denomina configuración y el conjunto de características seleccionadas representa una configuración del producto (Benavides et al., 2010; Apel et al., 2013). Un MC cumple con una estructura tipo árbol, y toda configuración siempre incluye la característica raíz (concepto de la LPS). La figura 1 (Apel y Kastner, 2009; Vidal et al., 2015) ilustra un ejemplo conocido de MC para un grafo.

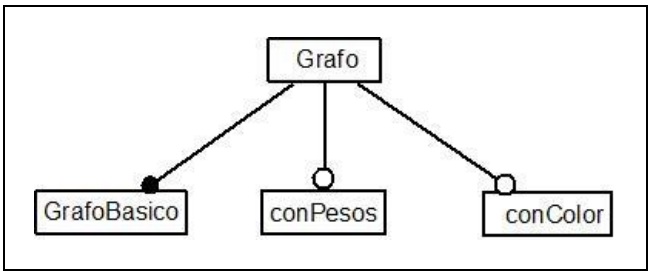

Fig. 1: Modelo de caracteristicas de la variabilidad de un grafo.

Según los trabajos de Apel y Kastner (2009), Benavides et al. (2010), Apel et al. (2013), los MC admiten las siguientes relaciones padre-hijo entre características: (i) Mandatoria (obligatoria): un círculo negro en el encabezado de la conexión sobre la característica hija (la característica hija es parte de la configuración cuando su padre es parte de esta y viceversa); (ii) Opcional: un círculo blanco en el encabezado de la conexión sobre la característica hija (la característica hija no es necesariamente parte de la configuración cuando su padre es parte de esta); (iii) Conjunto de características OR: un arco negro debajo de la función padre sobre el conjunto de características hijas, de las cuales al menos una de ellas debe estar en la configuración cuando el padre es parte de; y (iv) Conjunto de características XOR: un arco blanco debajo de la función padre sobre el conjunto de características hijas, de las cuales solo una de ellas debe estar en la configuración cuando su padre es parte de. Así, la figura 1 describe que un Grafo (concepto de la LPS) siempre incluye las características de un GrafoBasico (mandatoria), y opcionalmente las características conPesos (pesos en sus aristas) y conColor (color en sus vértices). Los MC también soportan Restricciones Cruzadas (RC) entre características tales como excludes (excluye) y requires (requiere) que se identifican gráficamente con una flecha doblemente dirigida entre las características asociadas y una flecha dirigida desde la característica de origen hasta la característica de destino (el origen requiere al destino), respectivamente (Apel y Kastner, 2009; Benavides et al., 2010; Apel et al., 2013).

FOP (Feature-Oriented Programming, siglas en Inglés de Programación Orientada a la Característica) representa de forma más simple y más directa de incumbencias cruzadas heterogéneas (Apel et al., 2006; Apel et al., 2013) que enfoques de modularización tradicionales como la Programación Orientada a Objetos (POO) y la programación modular. FOP trabaja sobre cambios de diferentes piezas de funcionalidad para los cuales la definición de puntos de unión no resulta en una simple tarea para su refinamiento. Además, FOP permite la definición conceptual de una LPS y sus productos mediante MC. Sin embargo, los MC no soportan una completa modularización de incumbencias cruzadas homogéneas ya que FOP presenta redundancias y repetición de rutinas en la aplicación de un mismo cambio en diferentes puntos de unión (Apel et al., 2006; Kästner et al., 2007; Apel y Kastner, 2009; Bošković et al., 2010; Apel et al., 2013; Zhang et al., 2014). Esto asunto de MC y FOP, en términos prácticos se traduce en soluciones no siempre completamente modulares por la posible existencia de módulos diferentes y que realizan una misma tarea. Como una propuesta de solución a la baja modularización de incumbencias cruzadas homogéneas en FOP y MC, existen soluciones para mezclar enfoques tradicionales de Programación Orientada a Aspectos (POA) estilo AspectJ (Eclipse, 2018) con FOP para así dar soporte a la modularización de incumbencias cruzadas homogéneas y heterogéneas (Apel et al., 2006; Kästner et al., 2007; Apel y Kastner, 2009; Bošković et al., 2010; Apel et al., 2013; Zhang et al., 2014; Tan et al., 2015). Sin embargo, sólo los trabajos de Bošković et al. (2010) y Tan et al. (2015) consideran agregar elementos propios de POA sobre MC.

Como AspectJ y los enfoques de POA tradicionales presentan dependencias implícitas entre las clases aconsejables (clases sobre las cuales los aspectos introducen o definen algún tipo de propiedad o comportamiento adicional) y los aspectos (funcionalidades usualmente cruzadas), las propuestas para combinar los enfoques de POA estilo AspectJ y FOP preservan estas dependencias implícitas entre

componentes de modularización (clases y aspectos), un problema para el desarrollo de soluciones de software modular mediante equipos de desarrollo independientes así como para el trabajo en la evolución de estas soluciones.

Para la modularización de incumbencias cruzadas homogéneas, Bodden (2011) y Bodden et al. (2014) presentan las Interfaces de Punto de Unión (JPI de las siglas en inglés de Join Point Interface) versus enfoques de POA tradicional estilo AspectJ. JPI permite definir interfaces de punto de unión entre clases y aspectos aconsejables; por lo tanto, las clases pueden exhibir interfaces de JPI y los aspectos implementan dichas interfaces. De esta forma, el enfoque JPI no utiliza dependencias implícitas entre los aspectos y las clases aconsejables, no hay más clases ingenuas (clases que no conocen el módulo que introdujo cambios sobre sus propiedades y/o comportamiento), ni existen dependencias de los aspectos con las clases aconsejables. Los trabajos de Vidal et al. (2014) y Vidal et al. (2017) presenta detalles de JPI para la creación de programas modulares y para la especificación formal de aplicaciones JPI, respectivamente. Así, desde un punto de vista práctico, el desarrollo de independiente de aplicaciones JPI con clases y aspectos modulares es de alta viabilidad (Bodden et al., 2014).

En la búsqueda de una producción masiva de software modular, para aprovechar las ventajas de FOP en la modularización de incumbencias cruzadas heterogéneas (Apel et al., 2006; Apel et al., 2013) y de JPI para modularizar incumbencias cruzadas homogéneas, este trabajo propone y aplica MC JPI para modelar características e incumbencias cruzadas y sus relaciones, para el modelamiento conceptual de una LPS como un primer paso para un completo proceso de desarrollo de software modular JPI + FOP. Justamente, los trabajos previos de Vidal et al. (2015), Vidal et al. (2015b) y Vidal et al. (2015c) ilustran ideas de una simbiosis JPI + FOP para el modelamiento de componentes de líneas de productos de software, pero no de MC, esto es, a nivel de diseño de productos de una LPS y no al nivel de diseño conceptual de MC y LPS. Como una propuesta de modelamiento FOP, Clafer (Antkiewicz et al., 2013) presenta un metamodelo para la combinación entre diagrama de clases y modelo funcional de un sistema de manera cruzada para agregar conocimiento de dominio en un contexto de modelo de características. Clafer ofrece herramientas para crear modelos y para la validación de modelos. Este artículo preserva el modelado conceptual de MC para LPS con la inclusión de conceptos de JPI, en la búsqueda de una simbiosis completa de JPI + FOP.

PRODUCCIÓN DE SOFTWARE MODULAR CON FOP, POA Y JPI

Esta sección describe los principales objetivos de los paradigmas FOP y POA con JPI como extensiones de la Programación Orientada a Objetos (POO) en la búsqueda de una producción de software modular (Stoiber et al., 2007; Vidal et al., 2014; Apel et al., 2013; Vidal et al., 2015).

FOP

FOP modulariza la colaboración de clases, también denominadas incumbencias cruzadas heterogéneas o de corte transversal, como características. Así, FOP permite el desarrollo paso a paso de las LPS (Stoiber et al., 2007; Apel et al., 2013; Vidal et al., 2015). Tal y como (Stoiber et al., 2007; Apel et al., 2013) indican, FOP modulariza de buena forma las incumbencias cruzadas estáticas: nuevos campos, métodos, clases y definiciones de interfaces. Sin embargo, según los trabajos de (Kästner et al., 2007; Apel et al., 2013), FOP carece de una modularización transversal adecuada para la evolución del software, ya que el software debe cambiar y adaptarse a modificaciones no predecibles. Particularmente, FOP no modulariza correctamente la repetición de código, una incumbencia cruzada homogénea reconocida (Vidal et al., 2015).

POA y JPI

La Programación Orientada a Aspectos, POA, propuesta por Kiczales et al. (1997), permite modularizar las funcionalidades o incumbencias cruzadas en POO como aspectos, por lo que los aspectos aconsejan las clases para agregar comportamiento y/o propiedades de manera estática y dinámica sobre las clases aconsejadas. Los aspectos se atienden (aconsejan) las clases de manera dinámica tal y como si fueran eventos. Sin embargo, según Bodden (2011) y Bodden et al. (2014), POA presenta dependencias implícitas entre las clases aconsejables y los aspectos. En primer lugar, los aspectos definen los puntos de corte en el comportamiento de las clases aconsejables; por lo tanto, las instancias de clases son completamente ajenas a la experimentación de posibles cambios de comportamiento y propiedades. En segundo lugar, los aspectos pueden ser espurios o no efectivos ante los cambios de firma en los métodos parte de los puntos de corte en los aspectos (fragilidad en los puntos de corte) (Bodden, 2011). Asimismo, (Bodden, 2011; Bodden et al., 2014) también indican que los módulos de aspecto estilo AspectJ, comprometen el desarrollo independiente del código base y los módulos de aspectos, ya que sus desarrolladores deben contactarse y presentar un conocimiento global sobre los módulos del programa (aspectos y clases del sistema). La figura 2 ilustra las dependencias entre clases y aspectos en POA tradicional.

Para aislar las incumbencias cruzadas y obtener programas modulares sin dependencias implícitas con POA, (Bodden, 2011) propone JPI para definir interfaces de punto de unión entre clases y aspectos, y así eliminar la asociación implícita de puntos de corte / consejo entre los aspectos y las clases aconsejables (Kiczales et al., 1997). Al igual que para POA tradicional, en aplicaciones JPI los aspectos representan incumbencias cruzadas sin una regla de punto de corte puntual. Los aspectos solo indican interfaces de puntos de unión para implementar. Además, para el código no ingenuo en JP, las clases aconsejables deben exhibir explícitamente instancias de JPI. La figura 3 ilustra las dependencias entre aspectos y clases en aplicaciones JPI. Aunque JPI y enfoques de POA tradicional como AspectJ modularizan adecuadamente las incumbencias cruzadas homogéneas, para la modularización de incumbencias cruzadas heterogéneas, estos enfoques no respetan la naturaleza de la POO y, a veces algunos de los principios básicos de POO tales como el ocultamiento de la información.

Teniendo en cuenta los principales beneficios de JPI y FOP, tal y como argumentan los trabajos de Vidal et al. (2015), Vidal et al. (2015b) y Vidal et al. (2015c), una simbiosis JPI y FOP parece muy adecuada. Siguiendo esta idea, la siguiente sección propone MC JPI en la búsqueda de un enfoque de desarrollo de software JPI + FOP completo que tenga en cuenta las principales ventajas y propiedades de FOP y POA con JPI para la producción de LPS modulares. Es importante destacar que los trabajos de Vidal et al. (2015), Vidal et al. (2015b) y Vidal et al. (2015c) ya propusieron y aplicaron el diagrama de colaboración JPI en un contexto JPI FOP, pero no en un contexto de la base conceptual de LPS con MC.

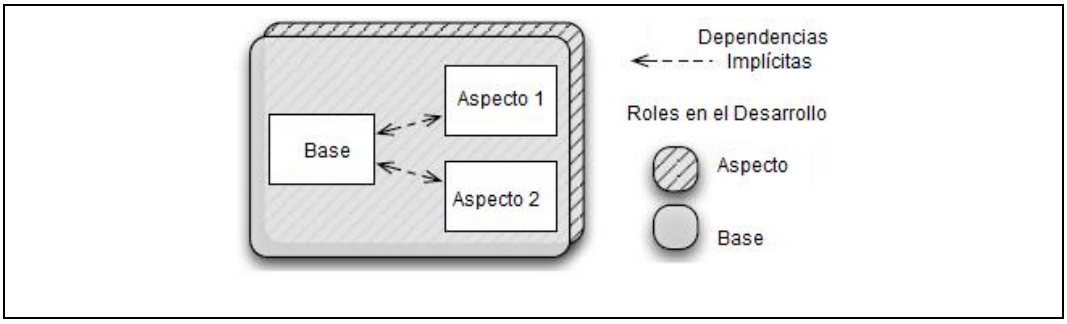

Fig. 2: Asociaciones entre módulo base y aspectos en POA tradicional.

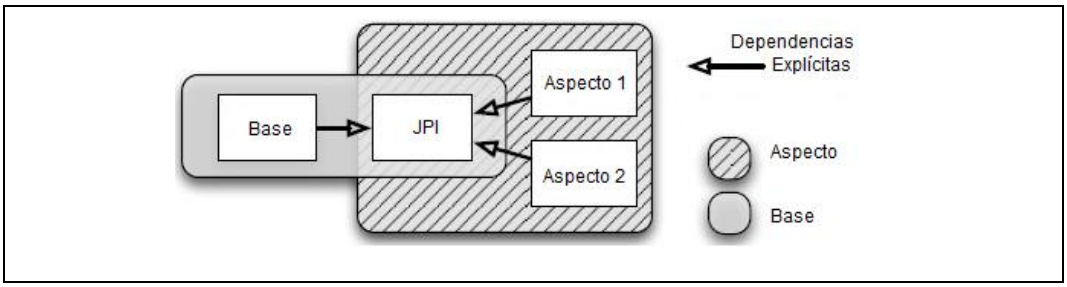

Fig. 3: Asociaciones entre modulo base y aspectos por medio de una interfaz de punto de unión en JPI

MC JPI: MODELO DE CARACTERÍSTICAS JPI

Para preservar los principios ya descritos de JPI y FOP, MC JPI amplía el modelo de características tradicional para admitir un nuevo tipo de característica JPI; por lo tanto, las características pueden exhibir una o más instancias de estas características. Las aristas de exhibición son aristas con un triángulo blanco en la característica JPI objetivo. Además, al igual que (Bošković et al., 2010), esta propuesta modulariza los modelos de características y distingue entre características de Incumbencias Base del Modelo de Características (IBMC) y características de Incumbencias Cruzadas del Modelo de Características (ICMC). Por lo tanto, las características IBMC pueden exhibir características ICMC. Además, para cada IBMC, es necesario definir reglas para la relación de exhibición que sería equivalente a una regla de punto de corte en POA tradicional. Así, en este trabajo no se requiere la definición de un modelo de características para las reglas de punto de corte. Para relacionar un ICMC con IBMC, el primero incluye una regla de implementación para definir la característica JPI que este implementa con detalles sobre el tipo de consejo y su momento de aparición (before o after) en el IBMC. Esto permite respetar parte de las ideas de POA tradicional junto con ideas de composición de JPI. De esta forma, un consejo de tipo alrededor (around) no sería de relevancia en un modelo conceptual como MC, ya que este tipo de consejo se puede expresar mediante una secuencia de consejos antes (before) y después (after). Por lo tanto, la asociación gráfica es útil para la documentación y como referencia para la definición de reglas.

Como una ICMC representa un modelo de característica, un ICMC puede exhibir otras instancias ICMC, en cuyos casos los modelos fuente se corresponden con los modelos de característica de las características base. Por lo tanto, la diferenciación entre ICMC e IBMC depende del contexto y las asociaciones de modelos actuales. En general, una instancia de MC puede definir reglas de implementación para vincularse usualmente a características externas, característica de otros módulos que le exhiben.

Reglas de Modelos de Características JPI

Las figuras 4 y 5 presentan detalles de reglas para exhibición e implementación de IBMC e ICMC, respectivamente. Es necesario considerar que JPIx representa una interfaz de punto de unión que las instancias de ICMC implementan e instancias tanto de IBMC e ICMC pueden exhibir. Téngase en cuenta que, para las reglas de exhibición, una característica aconsejable F puede conocer directamente sus hijos y el tipo de asociación entre F y sus hijos.

```
A exhibits JPIx: children(B) && children(C)
A exhibits JPIx: A.children(C) | | A.children(D)
A exhibits JPIx: children.Num>5 && A.children.Num < 10
A exhibits JPIx: XOR(B) && NOT A.OR(C)
A exhibits JPIx: children(B) && chosen(B)
```

Fig. 4: Regla de exhibición exhibits para instancias de IBMC.

```
B implements JPIx: before | after
```

Fig. 5: Regla de implementación de ICMC.

La figura 4 incluye detalles sobre los componentes para definir reglas de implementación. Por lo tanto, los conectores lógicos &&, II, NOT, => y <==> son utilizables para la conjunción, para la selección no exclusiva, para la negación, para la implicación y para la equivalencia, respectivamente. Como se aprecia en la figura 5, un modelo B que implementa una característica JPI solo necesita indicar un tipo de consejo, antes (before) o después (after). Así mismo, las reglas admiten el uso de operadores relacionales tales como >, <, = y su combinación. Además, las reglas soportan funciones para obtener elementos y propiedades del MC JPI analizado actualmente, tales como children (x) para saber si x es una característica hija de una característica citada o analizada y children.num para saber el número de características hijas de una característica actualmente analizada. Las reglas anteriores también se pueden definir para otras características identificadas.

Por ejemplo, para una característica f, f.children (x) es verdadero si una característica x es hija de f, f.children.num para obtener el número de hijos de f. Para conocer las propiedades de conjunto de características, XOR (f1) y OR (f1) son verdaderas si la característica f1 es parte de una asociación XOR o asociación OR para una característica madre analizada actualmente; y choosen (f) para saber si la característica f es parte de una configuración actual. Es importante destacar que también es posible utilizar f1.OR (f2) y f1.XOR (f2) para referir asociaciones en las que f1 es el padre y f2 es un hijo miembro de una asociación OR y XOR, respectivamente. La siguiente sección describe un caso de estudio que aplica esta propuesta. Como este ejemplo ya se modeló utilizando modelos de características orientadas a aspectos (Bošković et al, 2010), es posible resaltar los principales pros y contras de los MC JPI.

EJEMPLO DE APLICACIÓN DE MC JPI Y DISCUSIÓN DE RESULTADOS

Las figuras 6 y 7 (Bošković et al., 2010) muestran parte de un caso de estudio de un e-Shop ya modelado utilizando modelos de características orientadas a aspectos que este trabajo utiliza para aplicar MC JPI. En ambas figuras, la contraseña (password) representa una característica de incumbencia cruzada (aparece diferentes veces en el modelo original). Así, la figura 8 presenta el Modelo de Características JPI en el que la característica Administración exhibe la característica de incumbencia cruzada JPIAdm en el modelo e-Shop. De manera similar, la figura 9 presenta el modelo de características con una incumbencia cruzada donde la característica de registro (Register) exhibe JPIReg. Téngase en cuenta que las figuras 8 y 9 presentan instancias de IBMC que incluyen una regla de exhibición directa y simple (sin fórmulas). La figura 10 representa la ICMC que implementa JPIAdm mientras que la figura 11 ilustra el ICMC que implementa JPIReg. Cada figura de las instancias de ICMC regla simple, esto es, una asociación directa sin fórmulas. Las figuras 6, 7, 8, 9, 10 y 11 son diseñadas con FeatureIDE (Meinicke et al., 2017).

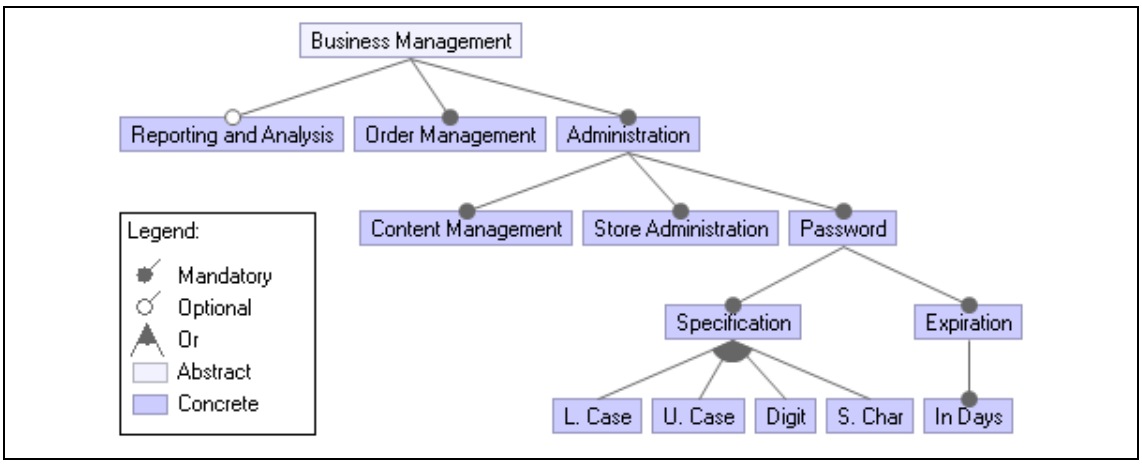

Fig. 6: Parte de modelo de característica de e-Shop (I).

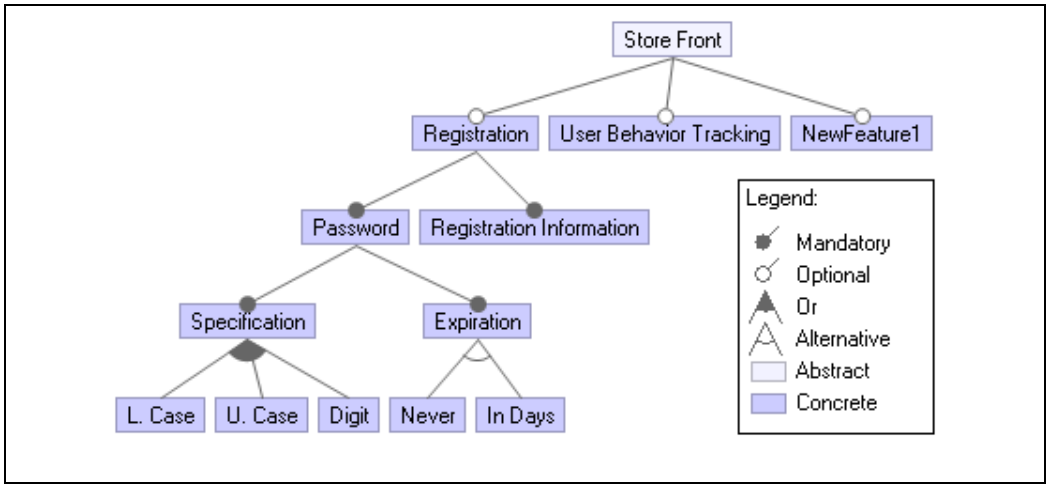

Fig. 7: Parte de modelo de característica de e-Shop (II).

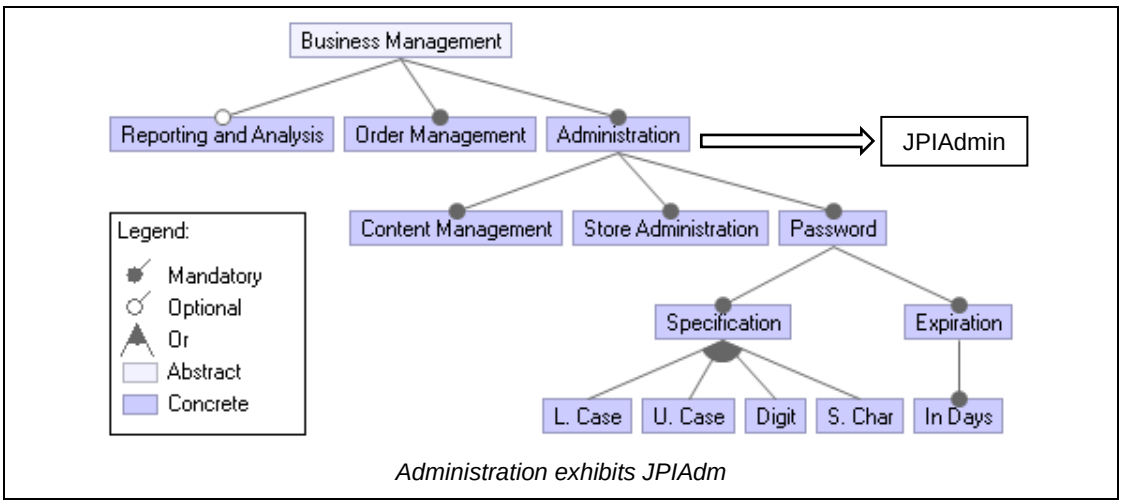

Fig. 8: IBMC del sistema e-Shop para la incumbencia de BackOffice.

Téngase en cuenta que JPI representa un enfoque de modularización para el cual la ambigüedad no está presente. Por esta razón, los componentes recomendados de un modelo de características deben exhibir modelo e-Shop, ya que esta propuesta solo aconseja en una situación, dicho ejemplo presenta solo un ICMC. Por lo tanto, para otros posibles consejos, para un comportamiento de ICMC diferente, se requieren nuevas instancias de ICMC. Este artículo presenta 2 diagramas de puntos de corte, uno para cada característica explícitamente aconsejable. Otra ventaja adicional de esta propuesta es la inclusión de 2 tipos de consejos (antes y después) respecto al trabajo de Bošković et al. (2010). Esta propuesta define reglas para exhibiciones e implementaciones, por lo tanto, las características aconsejables ya no son ingenuas.

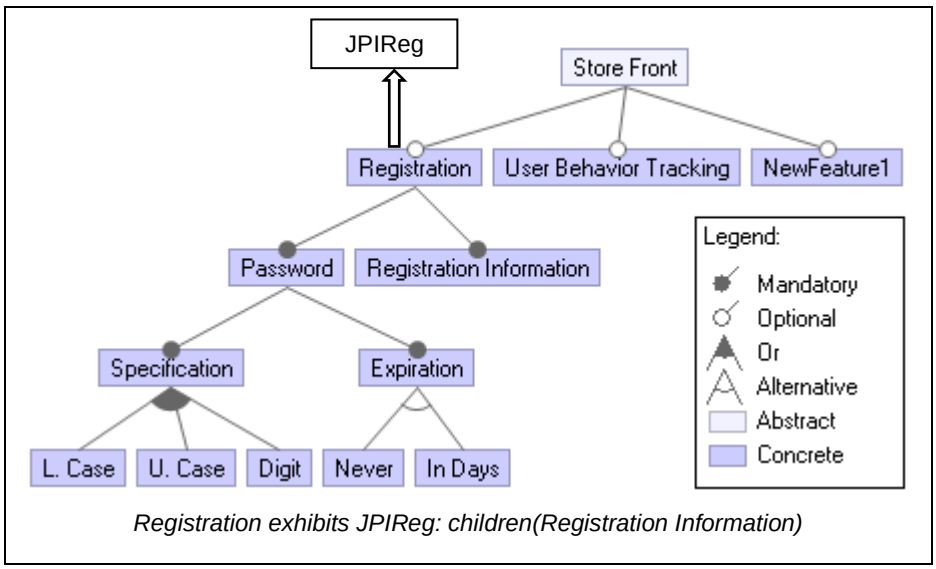

Fig. 9: IBMC del sistema e-Shop para la incumbencia de interfaz de usuario.

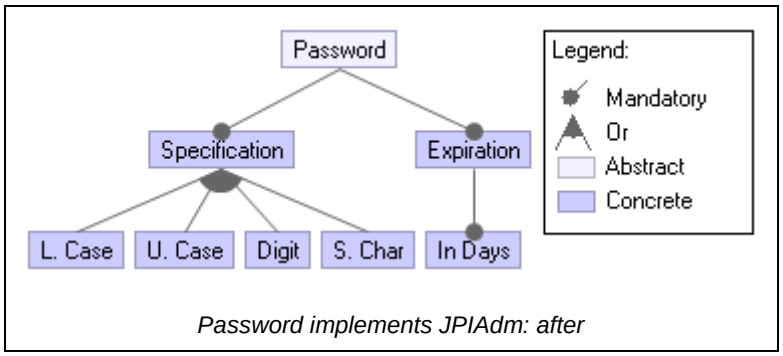

Fig. 10: ICMC del sistema e-Shop para la caracteristica password que implementa JPIReg.

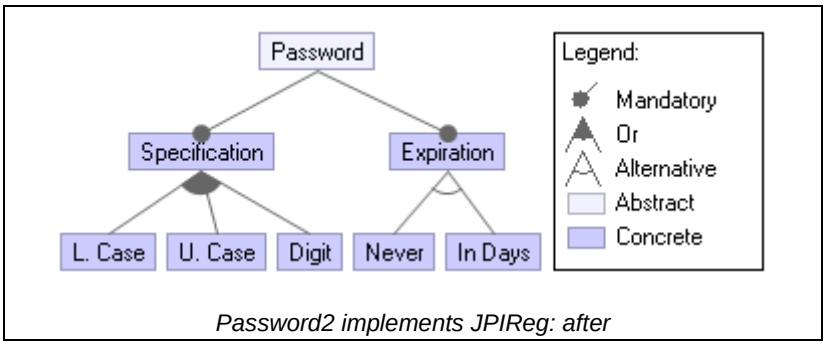

Fig. 11: ICMC del sistema e-Shop para la caracteristica password2 que implementa JPIAdmin.

Como parte de una investigación futura, se trabaja en el diseño de pruebas de aplicación de esta propuesta, y en la consideración de otros modelos y herramientas de modelamiento para dar soporte a la misma, con el objetivo de verificar su efectividad modular en casos de estudio reales. Además, se trabajará en el desarrollo de herramientas para la validación de modelos MC JPI.

CONCLUSIONES

Este trabajo propone la base conceptual de una nueva metodología de desarrollo de software modular, la simbiosis de los paradigmas FOP y AOP con JPI, la cual se aplica sobre casos de estudios tradicionales de ambos paradigmas para así demostrar la efectividad de la propuesta. Es importante resaltar que el MC JPI representa la base de una metodología de desarrollo de software JPI + FOP completo; esto es, para no cambiar de vista o enfoque en cada una de las etapas de su proceso de desarrollo. Claramente, desde el diseño del modelo de características, pensar en las características, los aspectos y las interfaces de puntos de

unión representan un requisito elevado. Como se menciona e ilustra en este artículo, JPI requiere definir nuevos CCFM para cada comportamiento aconsejable. Independientemente de este tema "conceptual", JPI permite eliminar el olvido, y parece prometedor para un enfoque de simbiosis JPI + FOP.

REFERENCIAS

Antkiewicz, M., K. Bak, A. Murashkin, R. Olaechea, J.H. Liang y K. Czarnecki, Clafer tools for product line engineering, doi: 10.1145/2499777.2499779, en Actas de 17th International Software Product Line Conference, SPLC'13, Tokio, Japan, 130-135 (2013)

Apel, S., D. Batory, C. Kastner y G. Saake, Feature-Oriented Software Product Lines, Springer (2013)

Apel, S. y C. Kastner, An Overview of¨ Feature-Oriented Software Development, doi: 10.5381/jot.2009.8.5.c5, Journal of Object Technology, 8(5), 49–84 (2009)

Apel, S., T. Leich y G. Saake, Aspectual Mixin Layers, doi: 10.1145/1134285.1134304, en Actas de 28th International Conference on Software Engineering ICSE, Shangai, China, 122-131, Mayo (2006)

Benavides, D., S. Segura y A. Ruiz-Cortés, Automated Analysis of Feature Model 20 Years Later, A Literature Review, doi: 10.1016/j.is.2010.01.001, Journal Information Systems, 35(6), 615-836 (2010)

Bodden, E., Closure Joinpoints: Block Joinpoints without Surprises, doi: 10.1145/1960275.1960291, en Actas de Tenth International Conference on Aspect-Oriented Software Development, AOSD '11, ACM, New York, NY, USA, 117-128 (2011)

Bodden, E., É. Tanter y M. Inostroza, Join point interfaces for safe and flexible decoupling of aspect, doi: 10.1145/2559933, ACM Transactions on Software Engineering and Methodology, 23 (1), 1-41 (2014)

Bošković, M., G. Mussbacher, E. Bagheri, D. Amyot, D. Gašević y M. Hatala, Aspect-Oriented Feature Models (Position Paper), en Actas de 15th International Workshop on Aspect-Oriented Modeling at 13th International Conference on Model Driven Engineering Languages and Systems, Oslo, Noruega (2010)

Eclipse, The AspectJ Project (2018)

Kästner, C., S. Apel y D. Batory, A Case Study Implementing Features Using AspectJ, en Actas de 11th International Software Product Line Conference, SPLC'07, Kyoto, Japón, 223-232 (2007)

Kiczales, G., J. Lamping y otros cinco autores, Aspect-Oriented Programming, ECOOP, SpringerVerlag (1997)

Meinicke, M., T. Thümb y otros cuatro autores, Mastering Software Variablity with FeatureIDE, Springer, ISBN: 978-3-319-61442-7 (2017)

Stoiber, R., S. Meier y M. Glinz, Visualizing Product Line Domain Variability by Aspect-Oriented Modeling, 2nd International Workshop on Requirements Engineering Visualization, REV 2007, New Delhi, India, Octubre (2007)

Tan, L., Y. Lin y H. Ye, An Aspect-Oriented Framework for Software Product Line Engineering, doi: 10.1145/2811681.2811703, en Actas de Australian Software Engineering Conference, Vol. III, Adelaide, Australia (2015)

Vidal, C., D. Benavides, J. A. Galindo y P. Leger, Exploring the Sinergies between Join Point Interfaces and Feature-Oriented Programming, en Actas de XX JISBD, Jornadas de Ingeniería de Software y Bases de Datos, Universidad de Cantabria, Santander, España, Septiembre (2015)

Vidal, C., D. Benavides, P. Leger, J. A. Galindo y H. Fukuda, Mixing of Join Point Interfaces and Feature-Oriented Programming for Modular Software Product Line, 10.4108/eai.3-12-2015.2262534, en Actas de BICT 2015, Nueya York, USA, Diciembre (2015b)

Vidal, C., D. Benavides, J. Á. Galindo y P. Leger, JPI Feature Model: Exploring a JPI and FOP Symbiosis for Software Modeling, doi: 10.1109/SCCC.2015.7416583, en Actas de Workshop on Advanced Software Engineering, Jornadas Chilenas de Computación, Santiago, Chile, Noviembre (2015c)

Vidal, C., M. Bustamante, M. Lappo y M. Nuñez, JPIAspectZ: Una Extensión de AspectZ para la Especificación Formal de Requerimientos de Aplicaciones Orientadas a Aspectos JPI, doi: 10.4067/S0718-07642017000600020, Información Tecnológica, 28 (6), 189-198 (2017)

Vidal, C., S. Rivero, R. Schmal y J. Morales, Revisión de un Enfoque Modular de Programación Orientada a Aspectos, doi: 10.4067/S0718-07642014000400020, Información Tecnológica, 25 (4), 185-192 (2014)

Zhang, Q., R. Khedri y J. Jaskolka, An Aspect-Oriented Language for Feature-Modeling, doi: 10.1007/s12652-013-0201-z, Journal of Ambient Intelligence and Humanized Computing, 5(3), 343-356, June (2014)