Información Tecnológica
Vol. 28(6), 189-198 (2017)
http://dx.doi.org/10.4067/S0718-07642017000600020

JPIAspectZ: Una Extensión de AspectZ para la Especificación Formal de Requerimientos de Aplicaciones Orientadas a Aspectos JPI

Cristian L. Vidal-Silva[(1)], Miguel A. Bustamante-Ubilla[(2,5)]*, María del Carmen Lapo-Maza[(3)], María de los Ángeles Nuñez-Lapo[(4)]
(1) Escuela de Ingeniería Informática, Facultad de Ingeniería, Ciencia y Tecnología, Universidad Bernardo O'Higgins, Avenida Viel 1497, Ruta 5 Sur, Santiago-Chile (e-mail: cristianvidal@docente.ubo.cl)
(2) Universidad de Talca, Fac. de Economía y Negocios, Av. Lircay s/n, Talca-Chile.
(e-mail: mabu@utalca.cl)
(3) Facultad de Ciencias Económicas y Administrativas, Universidad Católica de Santiago de Guayaquil, Av. Carlos Julio Arosemena Km. 1½ vía Daule, Guayaquil, Ecuador. (e-mail: maria.lapo@cu.ucsg.edu.ec)
(4) Refinería Esmeraldas EP Petroecuador. Km. 7 ½ Vía a Atacames Esmeraldas, Ecuador.
(e-mail: angelesnl20@gmail.com)
(5) Universidad Católica de Santiago de Guayaquil, Av. Carlos Julio Arosemena Km. 1½ vía Daule, Guayaquil, Ecuador.

* Autor a quien debe ser dirigida la correspondencia

Recibido Mar. 29, 2017; Aceptado Jun. 6, 2017; Versión final Jun. 30, 2017, Publicado Dic. 2017

Resumen

El objetivo de este artículo es proponer, describir y ejemplificar el uso de JPIAspectZ, una extensión del lenguaje formal orientado a aspectos AspectZ para la especificación formal de requerimientos de aplicaciones software. Considerando que las principales características de JPI son la definición de interfaces de punto unión, este articulo muestra como JPIAspectZ también soporta estas propiedades de JPI en un contexto de especificación formal de requerimientos. El Desarrollo de Software Orientado a Aspectos (DSOA), por medio de módulos de aspectos, permite solucionar un par de problemas de modularización del enfoque de Desarrollo de Software Orientado a Objetos (DSOO), pero el DSOA agrega dependencias implícitas entre clases y aspectos. La metodología DSOA-JPI define interfaces de puntos de unión entre artefactos aconsejables y aspectos aconsejadores y así solucionar los problemas de dependencia de DSOA tradicional estilo AspectJ para la producción de software modular. Se concluye que las interfaces de punto de unión como intermediarias entre clases y aspectos son perfectamente especificables formalmente con la propuesta de lenguaje JPIAspectZ.

Palabras clave: JPIAspectZ; JPI; AspectZ; Aspectos; Interfaz de Punto de Unión

JPIAspectZ: An Extension of AspectZ for Formal Requirement Specification of JPI Aspect-Oriented Applications

Abstract

The objective of this article is to propose, describe and exemplify the use of JPIAspectZ, an extension of the aspect-oriented formal language AspecrtZ for the formal requirements specification of software applications. Since the main JPI features are the join point interfaces definition, this article shows how JPIAspectZ also support these JPI properties in a formal requirements specification context. The Aspect-Oriented Software Development (AOSD), by mean of aspects modules, permits solving a few modularization issues of the Object-Oriented Software Development (OOSD) approach, but AOSD adds implicit dependencies between classes and aspects. The AOSD-JPI methodology defines join point interfaces between advisable artifacts and adviser aspects, thus solving the implicit dependencies of traditional AOSD AspectJ style for the modular software production. This article concludes that join point interfaces, as mediators between classes and aspects are perfectly specifiable in a formal way by the JPIAspectZ language proposal.

Keywords: JPIAspectZ; JPI; AspectZ; Aspects; Join Point Interface

INTRODUCCIÓN

El Desarrollo de Software Orientado a Aspectos (DSOA) permite la modularización de incumbencias cruzadas en etapas del Desarrollo de Software Orientado a Objetos (DSOO). Puesto que el DSOA nació en etapa de programación del DSOO, lograr una transparencia y consistencia de conceptos y diseño entre los artefactos para un total o completo DSOA parece muy compleja. Así, en la búsqueda de dicha transparencia, han existido diferentes propuestas de adaptaciones o extensiones de lenguajes de modelamiento para dar soporte al DSOA tales como diagramas de casos de uso UML orientados a aspectos (Jacobson y Ng, 2004) y diagramas de clases UML orientados a aspectos (Liu y Chuang, 2008). Específicamente, Wimmer et al. (2011) presenta un estudio de diagramas UML orientados a aspectos. Sin embargo, sólo existen un par de artículos acerca de lenguajes formales para la especificación de requerimientos de software orientados a aspectos; por ejemplo, Yu et al. (2005) y Vidal Silva et al. (2013) presentan AspectZ, (Vidal Silva et al., 2015; Vidal et al., 2013) describen OOAspectZ, y Nakajima y Tamai (2004), ilustran una versión orientada a aspectos de Alloy. Bodden et al. (2014) indica que, en soluciones de DSOA tradicional, existe una doble-dependencia entre los módulos base a ser aconsejados y los aspectos que los aconsejan tal y como muestra la Figura 1.

Para solucionar el asunto antes mencionado, tal como muestra la Figura 2, (Bodden et al., 2014; Bodden et al, 2011; Inostroza et al., 2011) proponen el uso de Interfaces de Punto de Unión (IPU – en inglés JPI) entre clases y aspectos.

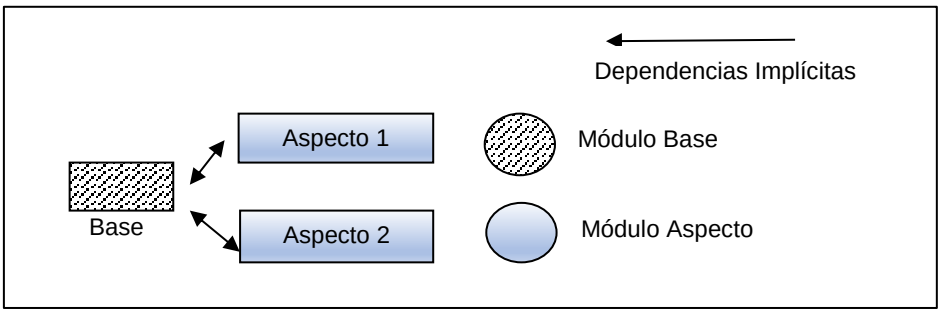

Fig. 1: Estructura de una Solución Orientada a Aspectos Estilo Aspect - (Adaptada de Bodden et al., 2014)

Considerando las ventajas conceptuales y prácticas de las soluciones JPI para la producción de software orientado a aspectos modular y, para lograr una transparencia de conceptos en todas las etapas o fases del proceso de Desarrollo de Software Orientado a Aspectos JPI (DSOA-JPI), este artículo propone y aplica JPIAspectZ, una extensión de OOAspectZ (Vidal Silva et al., 2015; Vidal et al., 2013) para la especificación formal de requerimientos de aplicaciones software JPI. Además, este artículo describe una lista de los pros y contras de JPIAspectZ. De esta forma, las principales contribuciones de este trabajo son: proponer lenguaje de modelamiento formal JPIAspectZ y presentar evidencia para mostrar la consistencia entre JPIAspectZ y otros artefactos de DSOA-JPI.

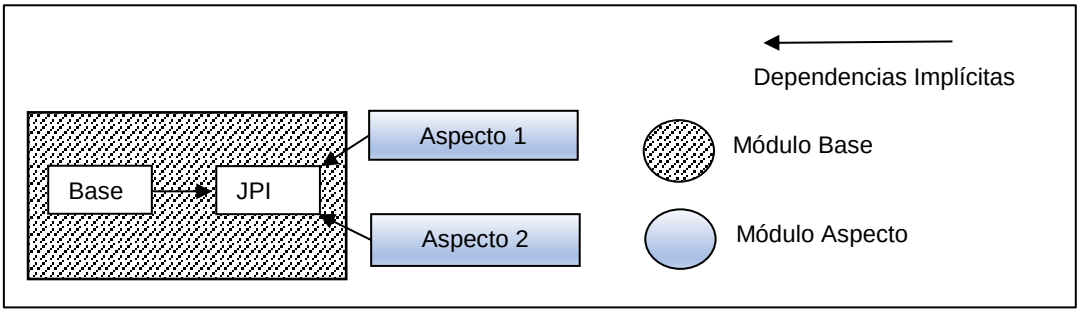

Fig. 2: Estructura de una Solución Orientada a Aspectos JPI - (Adaptada de Bodden et al., 2014)

Kiczales et al. (1997) y Kiczales (1996) proponen la Programación Orientada a Aspectos (POA), paradigma de programación para la modularización de incumbencias cruzadas en POO mediante el uso de módulos denominados aspectos. Así, los aspectos permiten, primero, modularizar funcionalidades y datos cruzados entre clases; y segundo, y aconsejar clases para la incorporación y uso de comportamiento y elementos estructurales tales como métodos y atributos en las clases ante la ocurrencia de eventos previamente definidos. Justamente, en los trabajos de Kiczales et al., (1997) y Kiczales, (1996), las clases son ingenuas de la existencia de los aspectos, así como de sus acciones.

Aun cuando, según Kiczales et al. (1997) y Kiczales (1996), POA permite solucionar asuntos de POO, esta metodología y paradigma de programación introduce problemas de modularización tal y como señala Bodden et al. (2014). Principalmente, POA introduce dependencias implícitas entre clases aconsejables y aspectos. Primero, los aspectos definen reglas de punto de corte (PCs) según el comportamiento de clases aconsejables; y, como resultado, las instancias de dichas clases son completamente ingenuas de posibles cambios en su comportamiento y estructura. Segundo, los aspectos pueden ser inefectivos o falsos dados los cambios posibles en la firma de métodos aconsejables de las clases objetivo.

Como señalan Bodden et al., (2014) y Bodden et al., (2011), este último asunto es conocido como problema del punto de corte frágil. Además, Bodden et al. (2014), también señala que, en POA clásica estilo AspectJ, e compromete el desarrollo independiente de módulos base y de aspectos puesto que los desarrolladores de módulos base y los desarrolladores de módulos de aspectos deben obtener un conocimiento global acerca de todos los componentes y módulos de un sistema, así como de las asociaciones entre ellos, esto es, existe una dependencia entre los equipos de desarrollo de módulos base y aspectos.

```
public class HelloWorld {

      public static void main(String[] args){
        say("Hello");
        sayToPerson("Hello", "Cristian");
    }

   public static void say(String message) {
        System.out.println(message);
    }

    public static void sayToPerson(String message,
                                   String name) {
        System.out.println(name + ", " + message);
    }
}
```

```
public aspect BasicAspect {
  pointcut callSayMessage():
     call(public static void HelloWorld.say*(..));

  before() : callSayMessage() {
     System.out.println("Good day!");
  }

  after() : callSayMessage() {
     System.out.println("Thank you!");
  }
}
```

```
package classes;

import joinpointinterfaces.*;
import aspects.*;

public class HelloWorld {

    /////JPIs//////////
    exhibits void JPISayBefore(): call(* say*(..));
    exhibits void JPISayAfter(): call(* say*(..));
    ////////////////////////////

    public static void main(String[] args){
       say("Hello");
       sayToPerson("Hello", "Cristian");
    }

   public static void say(String message){
      System.out.println(message);
    }

   public static void sayToPerson(String message,
                                  String name){
     System.out.println(name + ", " + message);
    }
}
```

```
package aspects;

import classes.*;
import joinpointinterfaces.*;

public aspect BasicAspect {
  before JPISayBefore(){
    System.out.println("Good day!");
  }

  after JPISayAfter(){
    System.out.println("Thank you!");
  }
}
```

```
package joinpointinterfaces;
import classes.*;

jpi void JPISayBefore();
jpi void JPISayAfter();
```

Fig. 3: Ejemplo de Clase HelloWorld y Aspecto BasicAspect en AspectJ (parte superior), y Solución JPI de Ejemplo de Clase HelloWorld, Aspecto BasicAspect y su Interfaz de Punto de Unión (parte inferior).

Actualmente, AspectJ es un ejemplo de lenguaje de programación de POA clásica (Griswold, et al., 2001). La parte superior de la Figura 3 muestra un ejemplo de aplicación AspectJ de elaboración propia. Claramente, la

clase HelloWorld es ingenua del aspecto BasicAspect, y dicho aspecto define una regla de punto de corte para aconsejar todas las llamadas a los métodos de la clase HelloWorld cuyo nombre comienzan con la palabra say. Así, si un método aconsejable cambia su nombre, para este ejemplo, si el nuevo nombre de alguno de los métodos que antes era aconsejable ya no se inicia con la palabra say, entonces el aspecto será inefectivo.

Para aislar incumbencias cruzadas y lograr programas POA modulares sin las mencionadas dependencias implícitas, el trabajo de Bodden et al. (2014) describe la metodología de programación JPI. JPI extiende la POA clásica permitiendo la definición de interfaces de punto de unión que las clases pueden exhibir y los aspectos pueden implementar. Tal como en POA clásica, según Bodden et al., (2014) y Bodden et al., (2011), en aplicaciones JPI, los aspectos representan funcionalidades cruzadas, pero sin la definición de PCs, ya que los aspectos sólo indican interfaces de punto de unión que ellos implementan. Además, en JPI las clases aconsejables ya no son ingenuas, ellas pueden exhibir explícitamente interfaces de punto de unión, esto es, las clases conocen acerca de los potenciales cambios que pueden ocurrir durante la ejecución de alguna de sus instancias.

La parte inferior de la Figura 3 muestra una solución JPI para el ejemplo de la clase HolaMundo y aspecto BasicAspect, con la definición de una interfaz de punto de corte para la definición su exhibición e implementación por la clase y el aspecto, respectivamente. Como se aprecia en la parte inferior de la Figura 3, una solución JPI define aspectos y clases no directamente enlazados, esto es, las clases y aspectos están relacionados indirectamente mediante el uso de interfaces de punto de unión. Justamente, la base de las soluciones JPI es definir una interfaz intermediaria entre clases aconsejables y aspectos. Claramente, las clases ya no son más ingenuas como lo eran en soluciones de POA clásica estilo AspectJ. Así, la parte superior de la Figura 3, muestra una solución AspectJ para una clase HelloWorld que es ingenua, con métodos estáticos say y sayToPerson, y un aspecto BasicAspect que aconseja estos métodos según la regla de punto de corte para la definición de puntos de unión para cuando alguno de sus consejos sea posiblemente efectivo. Por lo tanto, en la parte inferior de la Figura 3, la clase HelloWorld añade la definición de interfaz de punto de unión la cual exhibe, y el aspecto BasicAspect implementa dicha interfaz, lo que permite la eliminación de dependencias implícitas entre dichos componentes.

ASPECTZ, OOASPECTZ Y JPIASPECTZ

Z (Woodcock and Davies, 1996) y Object-Z (Smith, 2000) son lenguajes formales para la especificación de requerimientos de aplicaciones software. Específicamente, Z es un lenguaje de especificación formal clásico sin un soporte directo de abstracciones OO tales como clases y herencia, mientras Object-Z es una extensión de lenguaje Z para dar soporte a principios y abstracciones del DSOO. De igual manera, AspectZ (Vidal Silva et al., 2013; Yu et al., 2005;) y OOAspectZ (Vidal Silva et al., 2015; Vidal et al., 2013) representan extensiones de Z para la especificación formal de requerimientos de aplicaciones de POA junto con su integración con Z y Object-Z, respectivamente. Como Vidal et al. (2015) y Vidal et al. (2013) describen, OOAspectZ es una extensión de Object-Z (Duke, 2000; Smith, 2000; Smith, 1992) para soportar AspectZ (Yu et al., 2005) para la identificación y representación de requerimientos de incumbencias cruzadas en especificaciones Object-Z. Dado que la metodología de programación JPI es una importante mejora de POA para la producción de soluciones modulares, y en la búsqueda de lenguajes y herramientas consistentes para un completo DSOA-JPI, este trabajo propone y describe JPIAspectZ, una extensión de OOAspectZ para modelar aplicaciones JPI y su integración con Object-Z. En lo que sigue se presentan los componentes principales de especificaciones formales JPIApectZ: módulos base, interfaces de punto de unión y aspectos.

Módulos Base

A diferencia de AspectZ y OOAspectZ los que presentan módulos base ingenuos, los módulos de JPIAspectZ son similares a los módulos de clase de Object-Z, pero, además, los módulos aconsejables incluyen una regla de punto de corte exhibits para indicar el comportamiento aconsejable de la clase, es decir, una regla que define la ocurrencia de puntos de unión en la clase aconsejable.

Dado que la parte superior de un esquema de operación Object-Z permite definir sus parámetros de operación, en la búsqueda de una transparencia de conceptos y diseño en DSOA-JPI, una regla exhibits se compone de dos partes: promero, exhibits Interface donde interface se refiere a la interfaz de punto de unión que la clase exhibe, y segundo, un conjunto de condiciones para la ocurrencia del evento de punto de unión. En especificaciones JPIAspectZ, se consideran condiciones básicas de POA y JPO para cruzamiento dinámico y estático, esto es, llamadas de operaciones mediante call, ejecución de operaciones mediante execute, uso de conectores lógicos &&, ||, !, una función args para asociar y validar una lista argumentos de métodos a aconsejables, this (objeto) para identificar el objeto sobre el cual el método aconsejado se llama, y target (objeto) para identificar el objeto propietario del método aconsejable. En la siguiente sección se presenta una aplicación ejemplo de JPIAspectZ.

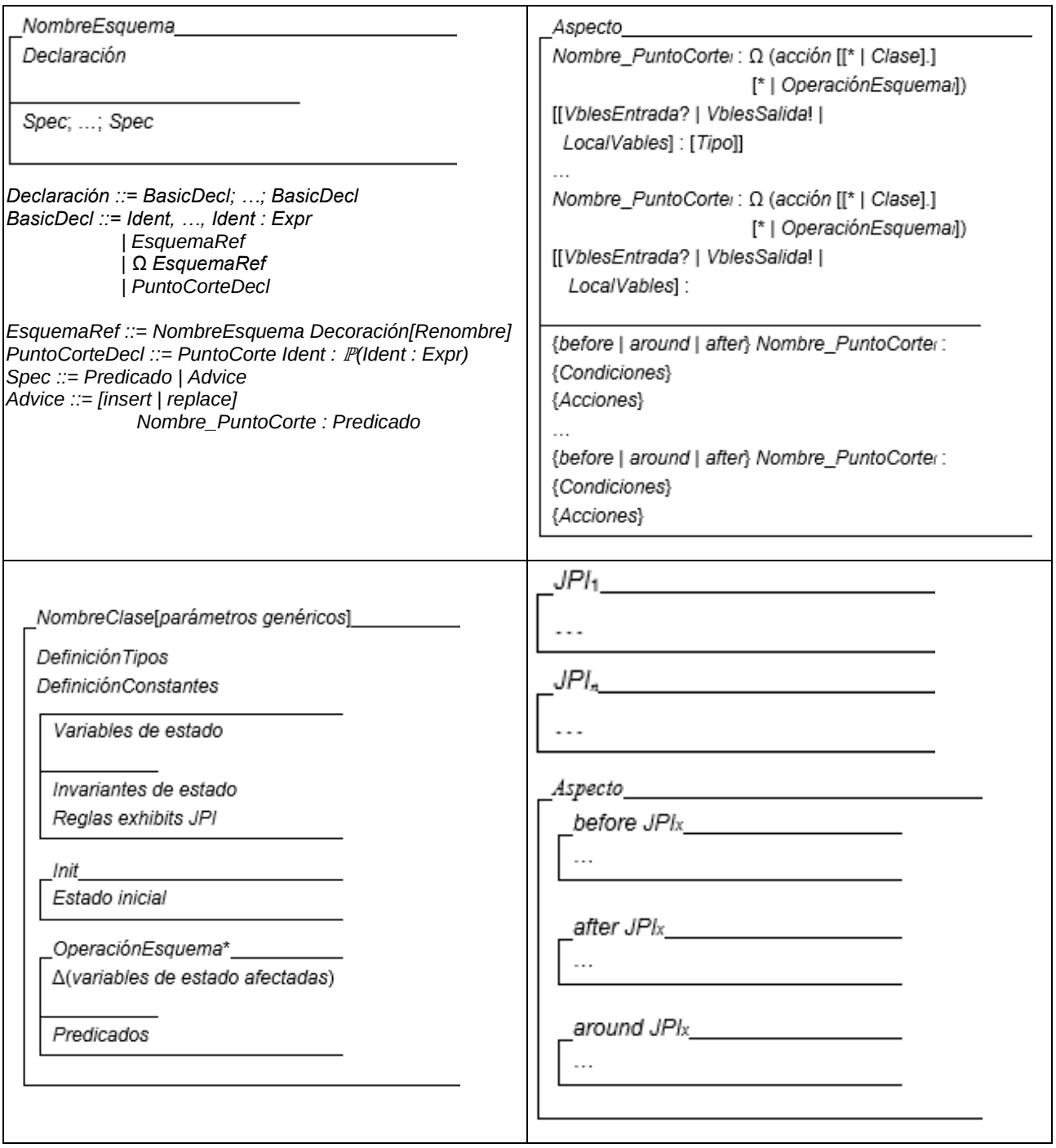

Fig. 4: Esquemas de Aspecto de AspectZ (superior derecha) y OOAspectZ (superior izquierda) (Adaptadas de Yu et al. (2005) y (Vidal Silva et al., 2015; Vidal et al., 2013) y Esquemas de Clase, JPI y Aspecto de una Especificación JPIAspectZ.

Se entiende como Interfaces de punto de unión, JPIAspectZ, la especificación de sistemas representada como esquemas de operación y que se denominan esquemas JPO. El nombre de los esquemas JPI se inicia con la palabra JPI. Por otra parte, Aspectos, JPIAspectZ, se representa por medio de esquemas de aspectos que son similares a esquemas de clase Object-Z. Así, un esquema de aspecto incluye, un esquema de estado, para definir atributos e invariantes, y esquemas de operación.

Como una diferenciación respecto a los esquemas de clase, los esquemas de aspectos permiten la inclusión de los términos before, after y around al inicio de los esquemas de operación para indicar tipos de consejo, esto es, el orden de acción de la operación del aspecto respecto a la acción aconsejada, la que provoca la ocurrencia de un punto de unión. Semánticamente, tal como Yu et al., (2005) presenta, los esquemas de aspecto aconsejan esquemas de operación, usualmente para la adición de comportamiento y atributos sobre las instancias de clase aconsejadas; OOAspectZ permite agregar comportamiento mediante consejos de tipo before, after y around sobre esquemas aconsejables.

La parte superior de la Figura 4 muestra esquemas de aspectos AspectZ y OOAspectZ, a la derecha e izquierda de la figura, respectivamente; mientras que la parte inferior de esta figura muestra la estructura de esquemas de clases JPIAspectZ, de esquemas de interfaz de punto de unión JPI, y de esquemas de aspectos. A partir de los esquemas de los métodos aconsejables, los que exhiben interfaces de punto de unión y los esquemas de aspectos que implementan dichas interfaces, JPIAspectZ, permite obtener esquemas tejidos. Esto permite destacar la evolución modular en los esquemas de los lenguajes JPIAspectZ, OOAspectZ, y JPIAspectZ.

APLICACIÓN DE JPIASPECTZ

El trabajo de Vidal et al., (2013) presenta el sistema de pintar (Painting System), que corresponde a un ejemplo de POA clásico que presenta las clases Point y Line (Punto y Línea) las cuales son figuras (Shape), y cada instancia de la clase Line está compuesta de un par de instancias de la clase Point. La idea principal de este ejemplo se ilustra en el proceso de actualizar la pantalla como un aspecto. La Figura 5 ilustra un diagrama de clases UML, el que presenta una interfaz Shape que clasifica a las clases Point y Line; y cada instance de Line está formada por dos instancias de Point, P1 y P2. Tal como se ha mencionado, en POA estilo AspectJ las clases son ingenuas a cualquier consejo, en este caso las clases Line y Point son ingenuas de las acciones del aspecto UpdateSignaling ya que las reglas de punto de corte no son parte de las clases.

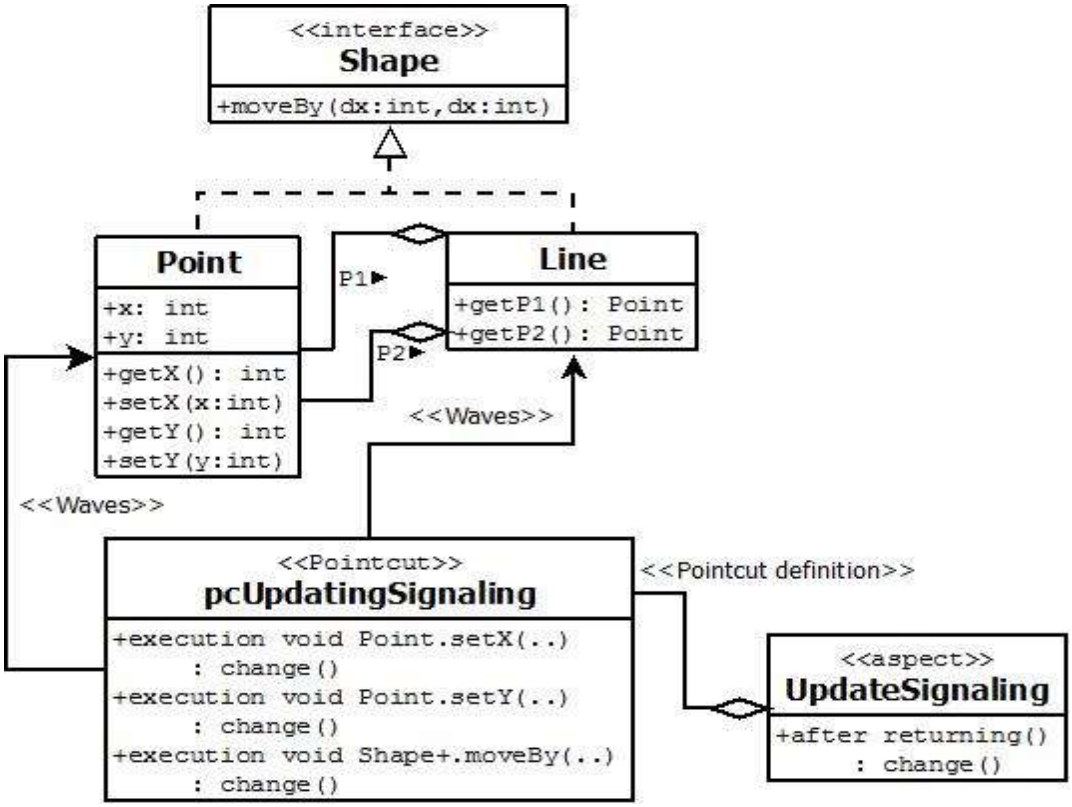

Fig. 5: Diagrama de Clases Orientado a Aspectos Estilo AspectJ de Sistema de Pintar.

La Figura 6 presenta un diagrama de clases UML JPI (Vidal Silva y Villarroel, 2014; Vidal Silva et al., 2014), del sistema de pintar con los siguientes componentes: una interfaz Shape; clases Point y Line; interfaces de punto de unión JPIUpdateX, JPIUpdateY, y JPIMove; y el aspecto Aspect1Painting. Así, las clases Point y Line exhiben las interfaces de punto de unión, específicamente la clase Point exhibe las interfaces JPIUpdateX y JPIUpdateY, y la clase Line exhibe la interfaz de punto de unión JPIMove, mientras que el aspecto Aspect1Painting implementa dichas interfaces de punto de unión.

La Figura 7 presenta una especificación formal JPIAspectZ del sistema de pintura. Claramente, esta especificación JPIAspectZ es completamente consistente con su diagrama de clases JPI de la Figura 6 para cada uno de sus componentes y la estructura de los mismos. Es relevante destacar, en consecuencia, que es posible lograr una consistencia en todas las etapas del DSOA-JPI. Por ejemplo, los trabajos de Bodden et al., (2014) e Inostroza et al., (2011) muestran un ejemplo de un sistema de comercio electrónico Shopping sesión, con una interfaz de punto de unión checkingOut, una clase ShoppingSession que exhibe la interfaz checkingOut, y un aspecto Discount que implementa dicha interfaz.

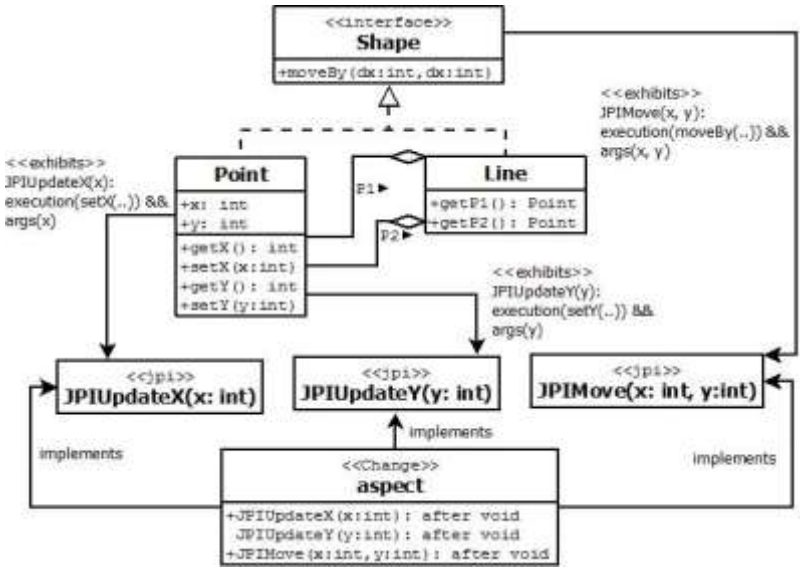

Fig. 6: Diagrama de Clases UML JPI de Sistema de Pintar.

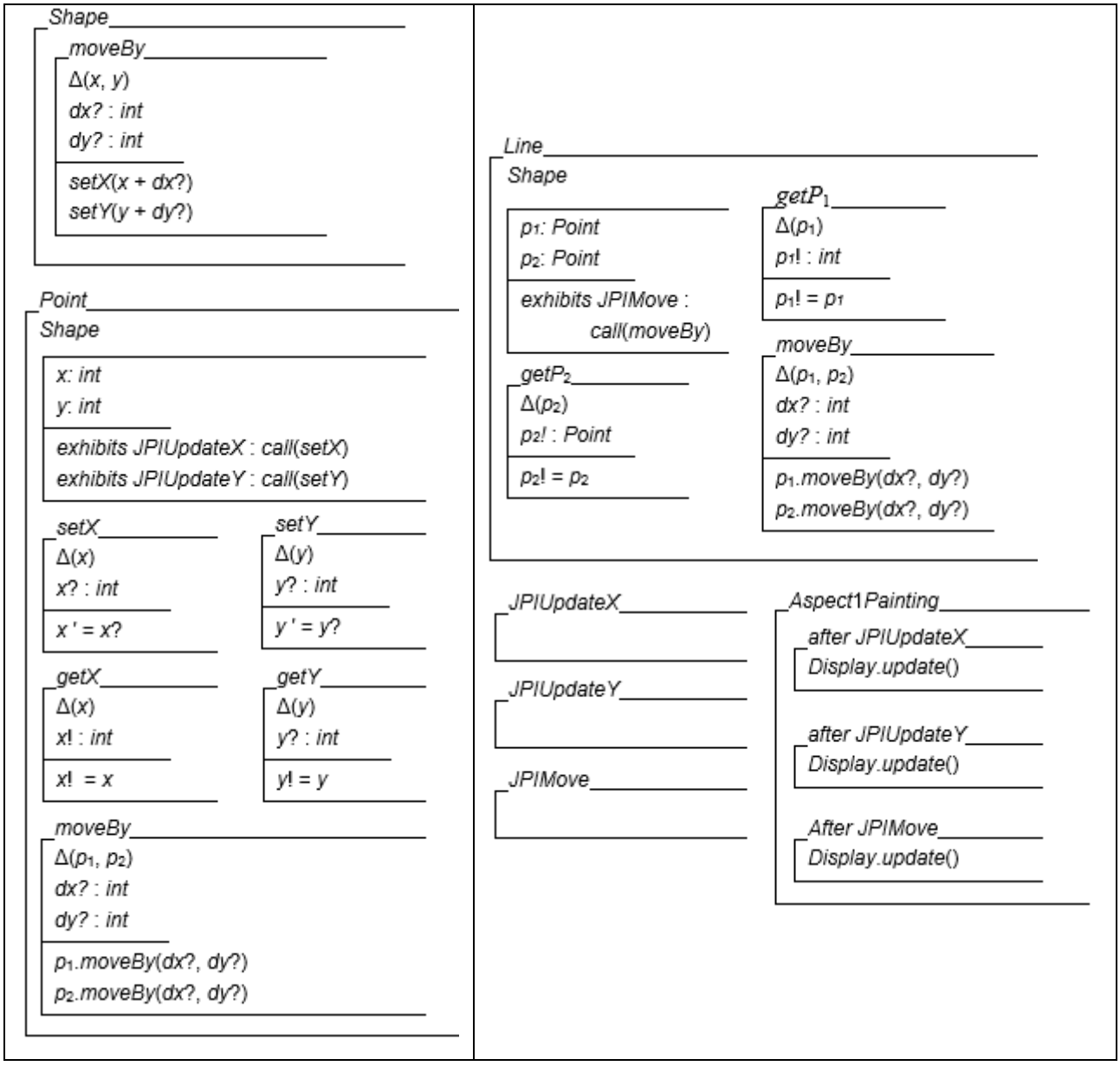

Fig. 7: Especificación JPIAspectZ de Sistema de Pintura.

La parte superior de la Figura 8, adaptada de Bodden et al., 2014 e Inostroza et al., 2011, muestra la especificación formal JPIAspectZ de este sistema, y la parte inferior que muestra el Código JPI de Sistema Shopping Session. Explicitando, nuevamente, una clara consistencia existente entre estos artefactos de DSOA-JPI.

Como un trabajo futuro, los autores proponen realizar investigaciones relacionadas con las extensiones a JPIAspectZ para el modelamiento de puntos de unión cerrados (closure join points) (Bodden, 2011), para el modelamiento de cruzamientos dinámicos avanzados (Apel et al., 2013), y para la simbiosis entre JPI y el modelamiento de características FOP (Vidal Silva et al., 2016). Además, para una validación automática de especificaciones JPIAspectZ, los autores proponen desarrollar una herramienta como parte de las herramientas CZT (CZT, 2017).

Fig. 8: Especificación JPIAspectZ de Sistema Shopping Session

CONCLUSIONES

Dados los avances de JPI como metodología de POA para la producción de software modular, este artículo presenta JPIAspectZ, un lenguaje formal para cubrir la brecha de especificación formal de requerimientos orientados a aspectos en el contexto de JPI.

JPIAspectZ permite especificar requerimientos de aplicaciones software JPI de manera formal, esto es, sin una dependencia implícita entre clases y aspectos lo que es consistente con la base de JPI. Además, JPIAspectZ, como una abstracción para el DSOA-JPI, demuestra una alta consistencia y transparencia de modelos y conceptos, específicamente una consistencia entre requerimientos y sus modelos estructurales, así como también entre requerimientos e implementación de código JPI. Probar una consistencia entre modelos estructurales y soluciones de código JPI parece directo.

Se confirma que, un lenguaje formal orientado a aspectos para el modelamiento de aplicaciones JPI similar a OOAspectZ permite razonar acerca de las principales propiedades, componentes y elementos de comportamiento de la metodología de POA JPI ya que existe un alto nivel de consistencia entre artefactos y modelos JPI. De esta forma, las clases aconsejables y las interfaces de punto de unión, como intermediarias entre clases y aspectos en quienes implementan dichas interfaces, son perfectamente especificables formalmente en el lenguaje JPIAspectZ de este trabajo.

REFERENCIAS

Apel, S., D. Batory, C. Kästner, y G. Saake, Feature-Oriented Software Product Lines: Concepts and Implementation, Springer Publishing Company Incorporated, 129-174 (2013)

Bodden, E., Closure Joinpoints, Block Joinpoints without Surprises, Proceedings of the 10th International Conference on Aspect-oriented Software Development, ACM, 117-128, Pernambuco, Brazil (2011)

Bodden, E., E. Tanter, y M. Inostroza, Join point Interfaces for Safe and Flexible Decoupling of Aspects, ACM Transactions on Software Engineering, 23(7), 7-41 (2014)

CZT: Community Z Tools: Tools for developing and reasoning about Z specifications. (En línea: http://czt.sourceforge.net/eclipse/, acceso: 30 de Septiembre (2016)

Duke, R. y R. Gordon, Formal Object-Oriented Specification Using Object-Z, Palgrave McMillan, UK (2000)

Griswold, B., E. Hilsdale, J. Hugunin, W. Isberg, y G. Kiczales, Aspect-Oriented Programming with AspectJ™. AspectJ.org, Xerox PARC, Tutorial (2001)

Inostroza, M., E. Tanter, y E. Bodden, Join Point Interfaces for Modular Reasoning in Aspect-Oriented Programs. Proceedings of ESEC/FSE '11, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, 508-511, Szeged, Hungry (2011)

Jacobson, I. y P. Ng, Aspect Oriented Software Development with Use Cases, Addison Wesley, 1st Ed., New York, USA (2004)

Kiczales, G., Aspect oriented programming. ACM Computing Surveys (CSUR) - Special issue: position statements on strategic directions in computing research, Volume 28, Issue 4es, Article N°154, New York, NY, USA, Dec. (1996)

Kiczales, G., J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, y J. Irwin, Aspect oriented programming. European Conference on Object-Oriented Programming (ECOOP), Springer-Verlag LNCS 124, 220-242, Finland (1997)

Liu, C. y C. Chuang-Weng, A State-Based Testing Approach for Aspect-Oriented Programming, Journal of Information Science and Engineering, 24, 11-31, Taipei, Taiwan (2008)

Nakajima, J. y T. Tamai, Lightweight Formal Analysis of Aspect-Oriented Models, Proceedings of the 5th Aspect-Oriented Modeling Workshop In Conjunction with UML, 120-127, Lisboa, Portugal (2004)

Smith, G., An Object-Oriented Approach to Formal Specification, Tesis Doctoral, Department of Computer Science, University of Queesland, Australia (1992)

Smith, G., The Object-Z Specification Language, Advances in Formal Methods, Springer-Verlag, Vol. 1, USA (2000)

Vidal Silva, C., R. Saens, R. Villarroel, y C. Del Rio, Aspect-Oriented Modeling: Applying Aspect-Oriented UML Use Cases and Extending AspectZ, Computing and Informatics, 32 (3), 573-593, Bratislava, Slovak (2013)

Vidal, C., R. Saens, C. Del Río, y R. Villarroel, OOAspectZ y diagramas de clase orientados a los aspectos para la Modelación Orientada a Aspectos (MSOA), Ingeniería e Investigación, 33 (3), 66 – 71, Medellín, Colombia (2013)

Vidal Silva, C. y R. Villarroel, JPI UML: UML Class and Sequence Diagrams proposal for Aspect-Oriented JPI Applications. XXXIII SCC International Conference of the Chilean Computer Science Society, Talca, Chile (2014)

Vidal Silva, C., S. Rivero, L. López, y C. Pereira, Propuesta y Aplicación de Diagramas de Clases UML JPI, Información Tecnológica, 25(5), 113-120, La Serena, Chile (2014)

Vidal Silva, C., R. Saens, R. Villarroel, C. Del Rio, y T. Tigero, Aspect-Oriented Formal Modeling: (AspectZ + Object-Z) = OOAspectZ, Computing and Informatics, 34(5), 996-1016, Bratislava, Slovak (2015)

Vidal Silva, C., J.A. Galindo, D. Benavides, R. Villarroel, P. Leger y S. Valenzuela, JPI Feature Models – Exploring a JPI and FOP symbiosis for software modeling, IEEE Explore, USA (2016)

Wimmer, M., A. Schauerhuber, G. Kappel, W. Retschitzegger, W. Schwinger, y E. Kapsammer, A Survey on UML-based Aspect-Oriented Design Modeling, ACM Computing Surveys, 43(4), 1-28 (2011)

Woodcock, J. y J. Davies, Using Z: Specification, Refinement, and Proof, Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1996)

Yu, H., D. Liu, L. Yang, y X. He, Formal Aspect-Oriented Modeling and Analysis by Aspect-Z, Proceedings of the 17th International Conference on Software Engineering & Knowledge Engineering, SEKE'2005, 124-132, Taipei, Taiwan, Republic of China, 14-16 July (2005)