Máquina de Aspectos: un enfoque alternativo para la implementación de aspectos.

Esteban Gesto, Karim Hallar y Sandra Casas

Universidad Nacional de la Patagonia Austral. Unidad Académica Río Gallegos.
Lisandro de la Torre 1070. CP 9400. Río Gallegos. Santa Cruz. Argentina
Tel/Fax: +54-2966-442313/17.
E-mail: lis@uarg.unpa.edu.ar

RESUMEN

La Programación Orientada a Aspectos (POA) es un nuevo paradigma de programación que propone mecanismos para soportar la separación de los aspectos no funcionales de los sistemas software. Una herramienta POA consiste en tres componentes principales: un lenguaje de programación base, un lenguaje de programación orientado a aspectos y un tejedor de aspectos (weaver). El presente trabajo propone un enfoque alternativo para la implementación de aspectos basado en la especificación de los mismos y sus elementos.

Palabras claves: Programación Orientada a Aspectos, Tejedor de Aspectos.

1. INTRODUCCIÓN

La mayoría de los sistemas software tienen propiedades que no necesariamente se alinean con los componentes funcionales de un sistema. Manejo de fallas, persistencia, comunicación, replicación, coordinación, registro de actividades, logging, traza, etc., son aspectos del comportamiento de un sistema que tienden a cortar transversalmente a grupos de componentes funcionales. Aunque sea posible analizar estos aspectos por separado de la funcionalidad básica de una aplicación, su implementación, usando los actuales lenguajes orientados a componentes, tiene como resultado que el código de estos aspectos se repite y esparce en muchos componentes funcionales. En consecuencia, el código fuente se convierte en una "maraña" o enredo de instrucciones de diferentes propósitos.[1]

La POA [2] es un nuevo paradigma de programación que aspira a soportar la separación de los aspectos antes mencionados. Es decir, la POA intenta separar los componentes (clases, métodos y/o objetos de funcionalidad básica) y los aspectos unos de otros, proporcionando mecanismos que posibiliten abstraerlos y componerlos para formar todo el sistema.

Una herramienta POA consiste de tres componentes principales: un lenguaje de programación base, un lenguaje de programación orientado a aspectos (LOA) y un tejedor de aspectos (weaver) [3].

El presente trabajo propone un enfoque alternativo para la implementacion de aspectos que prescinde del lenguaje de programación orientado a aspectos y permite aplicar los conceptos principales de la POA y facilitar el aprendizaje de los mismos.

2. LENGUAJES DE PROGRAMACION ORIENTADOS ASPECTOS

Los LOA son extensiones de lenguajes de programación convencionales que definen e incorporan los mecanismos necesarios para dar soporte a los aspectos. Estos mecanismos permiten implementar los cortes de una manera clara, proporcionando constructores que describan la semántica y el comportamiento de los aspectos [2].

Los principales constructores que en general incorporan los LOA dan soporte a los siguientes conceptos básicos del paradigma:

Puntos de unión (join points) : un punto de unión o de enlace es la ubicación que es afectada por el aspecto [4]. Es el lugar donde el tejedor introduce el código del aspecto. Los puntos de unión pueden estar presentes tanto en el ámbito de declaración como en el de operación. En el primero implica que el conjunto de puntos de unión posibles incluye cada declaración (línea de código) en el sistema, mientras que el segundo implica que este conjunto incluye cada operación (invocación de métodos) que el sistema realiza.

Puntos de Corte (pointcuts) : Los puntos de corte son elementos conceptuales que indican lugares bien definidos dentro del flujo del programa donde se puede realizar la composición de un aspecto. Por lo general los puntos de corte cortan los siguientes puntos de unión:

- métodos
- constructores
- manejadores de excepciones
- atributos

Aspectos: En la mayoría de los LOA, es posible definir entidades de primera clase que representen a los aspectos. Esta construcción, es bien una clase o una entidad muy parecida a una clase, en esencia es una unidad de código con un nombre y con variables y métodos propios.

En general, los aspectos cortan los componentes de funcionalidad básica para comunicar que cierto código se ejecutará antes, después o durante la ejecución de un evento (método, excepción, acceso a atributos, etc.); introducir atributos y métodos; modificar la estructura jerárquica; etc.

Estas características se pueden encontrar en AspectJ [5][6], y todos aquellos LOA que se basan en este tipo de modelo, como AspectS [7][8], AspectC++ [9][10], AspectC [11], Aurelia[12], Pythius [13], AspectR [14], etc.

A continuación se presenta un enfoque que sustituirá a un LOA, denominado Máquina de Aspectos, para el desarrollo de aspectos para aplicaciones cuyo lenguaje base es Java.

3. MAQUINA DE ASPECTOS

La propuesta radica en el diseño e implementación de una herramienta que permita la especificación de los aspectos de manera tal que el desarrollador no se encuentre ante la necesidad de aprender un nuevo lenguaje de programación. A esta herramienta se la denomina "Máquina de Aspectos" dado que sirve para definir aspectos.

La Máquina de Aspectos al igual que un LOA satisface las siguientes propiedades definidas en [15]:

- cada aspecto debe ser claramente identificable;
- cada aspecto debe autocontenerse;
- los aspectos deben ser fácilmente intercambiables;
- los aspectos no deben interferir entre ellos;
- los aspectos no deben interferir con los mecanismos usados para definir y evolucionar la funcionalidad, como la herencia.

Algunas directrices para la especificación y diseño de lenguajes de aspectos se observan en [16] y se aplican a la Máquina de Aspectos :

- la sintaxis está relacionada con la sintaxis del lenguaje base;
- el lenguaje debe ser diseñado para especificar aspectos en una forma concisa y compacta;
- la gramática del lenguaje debe tener elementos para permitir la composición de clases y aspectos.

3.1 Diseño de la Máquina de Aspectos

Un diseño simplificado de la Máquina de Aspectos se ilustra en el diagrama de clases de la Figura 1. En este diagrama se representa que un proyecto de software orientado a aspectos (Project) se compone de aspectos (Aspect). Un aspecto se conforma de elementos de aspectos (AspectElement). Existen tres tipos de elementos de aspectos: Métodos (Method) Atributos (Atribute) y avisos (Advice). La especificación de un aspecto y sus elementos se construye como una colección de objetos persistentes.

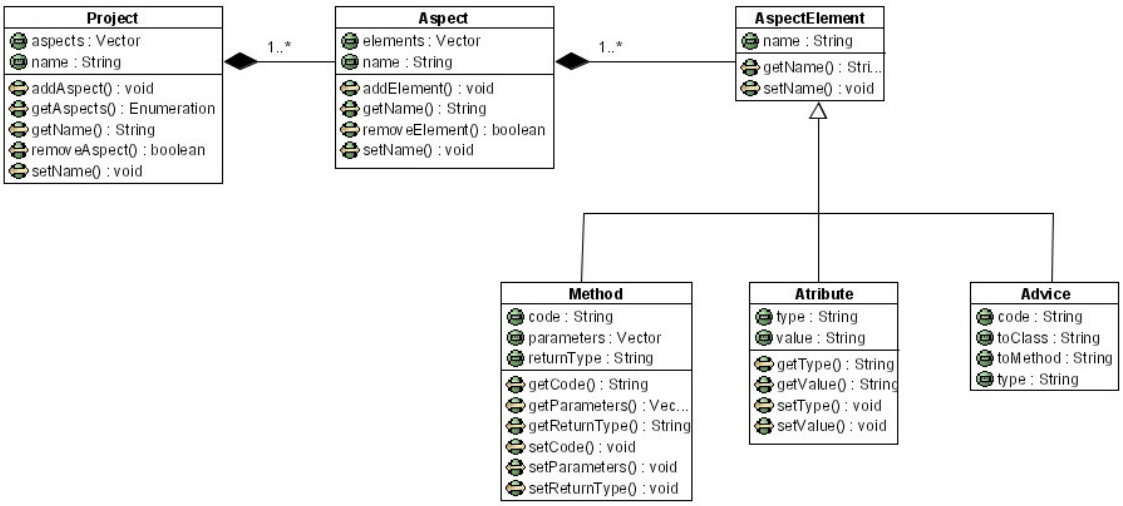

Figura 1: Diagrama de Clases Simplificado de la Máquina de Aspectos

La especificación de aspectos resulta muy sencilla para el programador, si conoce el lenguaje Java, puesto que es la única condición que se requiere. La especificación de atributos requiere los siguientes parámetros: nombre, tipo y valor inicial. La especificación de los métodos requiere nombre, tipo de devolución, parámetros y el código. En la especificación del aviso se establece el nombre del mismo, punto de unión (clase y método), tipo de aviso y código. Como parte del código de un aviso se pueden enviar mensajes a los métodos del aspecto o hacer referencia a los atributos del mismo.

En la Figura 2 se proporciona un ejemplo de la Máquina de Aspectos. Se ha escogido una interfaz gráfica e interactiva ya que facilitará su aprendizaje y uso. Como se puede observar, la pantalla se divide

en dos paneles principales. En el panel izquierdo, de manera jerárquica se ubican los aspectos y sus elementos en forma distintiva, correspondientes al actual proyecto, a medida que se van creando. En el panel derecho se disponen los formularios para la creación, eliminación o modificación de los aspectos y sus elementos.

La definición interactiva de los aspectos y sus elementos permiten realizar durante la especificación un conjunto de validaciones que facilitan la detección de ciertos errores cometidos por el desarrollador rápidamente, por ejemplo la duplicación de nombres o avisos relacionados a puntos de unión inexistentes. De otro modo, este tipo de errores se hallarían en tiempo de compilación o ejecución.

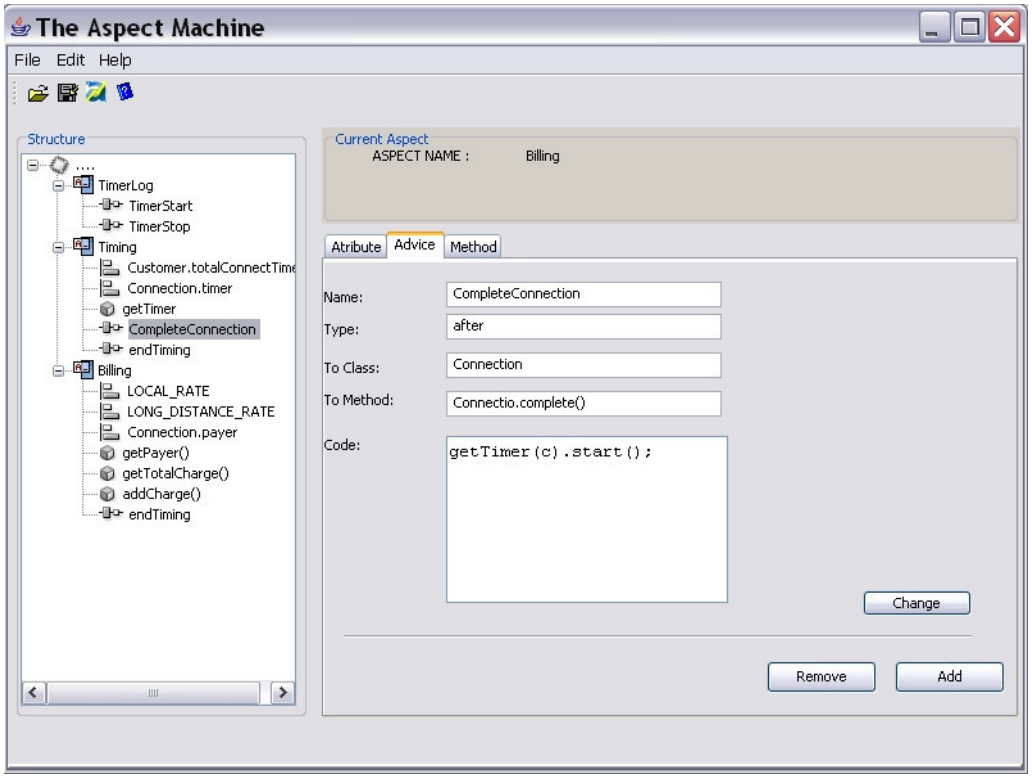

Figura 2: Interfaz Principal de la Máquina de Aspectos

3.2. Tejedor de Aspectos

El tejedor de aspectos es el proceso que tiene el propósito de componer, integrar o mezclar el código funcional con el código de los aspectos generando código ejecutable. Un tejedor de aspectos y clases utiliza técnicas de transformación de programas. No obstante, el interrogante es cuándo o en qué momento se produce el proceso de tejido. El momento en el cual ocurre el proceso de tejido constituye una de las principales diferencias entre los LOA. Si el tejido ocurre antes de la ejecución, es decir, en tiempo de compilación, es usualmente denominado tejido estático y si el tejido ocurre durante la ejecución, se lo conoce como tejido dinámico. Un tercer caso es el tejido en tiempo de carga (Load-Time), consiste esencialmente en aplicar técnicas basadas en la transformación de código a nivel binario. Este tipo de transformación post-compilación, es empleada principalmente en lenguajes que, como Java, generan un código intermedio denominado byte-code, el cual retiene una gran cantidad de información semántica. Este mecanismo consiste en modificar el byte-code en el momento en que se cargan las clases para su ejecución, lo cual requiere modificar el cargador de clases (class loader). Existen varias herramientas que soportan este mecanismo, por ejemplo JOIE [17], Javassist [18] y BCA [19] permiten reescribir el código para su modificación "on-the-fly". Este tipo de tejido tiene características deseables como no requerir recompilación de toda la aplicación ante modificaciones locales y que el código resultante de las aplicaciones no es invadido por las modificaciones añadidas.

Aplicar esta estrategia para el proceso de tejido de la Máquina de Aspectos implica contar con un componente de software cuya responsabilidad consista en cargar las clases para la posterior interpretación de la JVM (Java Virtual Machine). Un cargador de clases personalizado permitirá detectar previamente las clases que son afectadas por los avisos definidos en los aspectos y, en consecuencia, requieren ser transformadas.

4. CONCLUSIONES

El presente trabajo propone un enfoque para el desarrollo de aplicaciones orientadas a aspectos, basado en la especificación de aspectos y sus elementos. La especificación se desarrolla mediante una herramienta visual e interactiva que se ha denominado Máquina de Aspectos, por la cual los distintos elementos de la especificación se crean como una colección de objetos, de manera simple y transparente para el programador.

El primer objetivo del trabajo consiste en ensayar y probar los conceptos básicos del paradigma como ser cortes y avisos para más tarde adicionar características más complejas como herencia de aspectos, detección de conflictos, etc.

El trabajo actual y futuro está centrado en finalizar la implementación del tejedor de aspectos y probar su comportamiento para luego extender hacia nuevas propiedades.

El presente trabajo fue parcialmente financiado por la Universidad Nacional de la Patagonia Austral, Santa Cruz, Argentina. Este trabajo forma parte del proyecto de investigación "Estrategias para la Resolución de Conflictos en AspectJ", radicado en la UNPA.

REFERENCIAS

[1] Hursch W., Lopes C., "Separation of Concern". Tech. Rep. NU-CCS-95-03, Northeastern University, 1.995.

[2] Kiczales G., Lamping L., Mendhekar A., Maeda C., Lopes C., Loingtier J., Irwin J., "Aspect-Oriented Programming". In Proceedings ECOOP'97 – Object-Oriented Programming, 11th European Conference, Jyväskylä Finland, Springer-Verlang, 1.997.

[3] Piveta E., Zancanela L., "Aspect Weaving Strategies", Journal of Universal Computer Science, vol.9, no. 8, 2.003.

[4] Ossher H., Tarr P., "Multi-Dimensional Separation of Concerns and the Hyperspace Approach", in Proceedings of the Symposium on Software Arquitectures and Component Technology: The state of the Art in Software Development. Kluwer, 2.001.

[5] Kiczales G., Hilsdale E., Hugunin J., Kersten M. Palm J., Griswold W. "An Overview of AspectJ". ECOOP 2.001.

[6] Homepage de AspectJ Xerox, PARC, USA http://aspectj.org/ .

[7] Hirschfeld R., "AspectS – AOP with Squeak", in Proceedings of OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented System, 2.001.

[8] Homepage de AspectS http://www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/

[9] Gal A., Schroder W., Spinczyk O., "AspectC++: Language Proposal and Prototype Implementation", ACM International Conference Proceeding Series Proceedings of the Fortieth International Conference on Tools Pacific. Vol.10. Australia. 2.002.

[10] Homepage de AspectC++, http://www.aspectc.org/.

[11] Homepage de AspectC: http://www.cs.ubc.ca/labs/spl/projects/aspectc.html

[12] Piveta E., Zancanela L. "Aurelia: Aspect oriented programming using reflective approach". Workshop on Advanced Separation of Concers ECOOP. 2.001.

[13] Homepage de Pythius: http://sourceforge.net/projects/pythius/

[14] Homepage de AspectR: http://aspectr.sourceforge.net/

[15] Cugola G., Ghezzi C., Monga M., "Coding Different Design Paradigms for Distributed Applications with Aspect-Oriented Programming", in the Workshop su Sistemi distribuiti: Algoritmi, Architetture e Linguaggi (WSDAAL). 1.999.

[16] Boellert, K., "On Weaving Aspects", Proc. of the AOP Workshop at ECOOP 1.999.

[17] Keller R., Holzle U., "Binary Component Adaptation". ECOOP 1998, Object Oriented Programming, vol. 1445, pp. 307-329. Springer, 1.998.

[18] Chiva S., "Javassist – A Reflection – based Programming Wizard for Java", In Proceeding of the ACM OOSPLA´98 Workshop on Reflective Programming in C++ and Java.1998.

[19] Cohen G., Chase J., Kaminsky D., "Automatic Program Transformation with JOIE". Proc. of the USENIX, Anual Technical Conference, pp. 167-178, USA. 1998.