Investigación y Tecnología Vol. 3(1-2): 86-97. Diciembre 2015. ISSN 2306-0522

Usabilidad de la programación Orientada a Aspectos

Usability Aspect Oriented Programming

***Nancy Orihuela Sequeiros, Rubén Alcon López**
Instituto de Investigaciones en Informática
Carrera de Informática
Facultad de Ciencias Puras y Naturales
Universidad Mayor de San Andrés
La Paz - Bolivia
Autor de correspondencia:* nancyori@hotmail.com**,** ralconlo@gmail.com

Resumen

El presente proyecto propone el estudio y análisis del paradigma de programación orientado a aspectos (POA); desde la definición de lo que es un aspecto hasta el estudio de su incursión dentro de los contenidos curriculares de la Carrera de Informática. En el proceso de investigación se ha establecido el conjunto de atributos que ofrece el paradigma orientado a aspectos; desde la necesidad de un nuevo enfoque en la fase de análisis y definición de requerimientos hasta la necesidad de incorporar un conjunto de nuevas herramientas que muestren de manera clara y simple la inclusión de rutinas "especiales" que coadyuven a mejorar el producto software sin entremezclar código de las funcionalidades básicas y las funcionalidades especiales. En este nuevo enfoque se ve la necesidad de disponer de un conjunto de herramientas que hagan posible los resultados mencionados y se plantea la necesidad de disponer de Lenguajes de programación orientados a aspectos (LPOA) y lenguajes de descripción arquitectónica adecuados a este paradigma. Además se considera importante disponer de un conjunto de métricas que valoren la calidad de los productos obtenidos con este nuevo paradigma a fin de que los mismos cumplan con algunas de las condiciones impuestas por los estándares existentes.

Palabras clave: Paradigma Orientado a Aspectos (POA), Programación Orientada a Aspectos, Aspectos (LPOA), LDA.

Abstract

This project proposes the study and analysis of the paradigm of aspect-oriented programming (AOP); from the definition of what is an aspect to study their foray into the curriculum of the School of Computing. In the process of research has established the set of attributes offered by the aspect-oriented paradigm; from the need for a new approach in the phase of analysis and requirements definition to the need to incorporate a set of new tools that show a clear and simple way to include "special" routines that help to improve the product software code without intermixing of the basic functions and special features.

In this new approach is the need to have a set of tools that enable the above results and the need for programming languages oriented aspects (LPOA) and architectural description languages suitable for this paradigm arises. Also it considered important to have a set of metrics to assess the quality of the products obtained with this new paradigm so that they fulfill some of the conditions imposed by existing standards.

***Keywords**: Aspect oriented paradigm (POA), Aspect Oriented Programming (LPOA), LDA.*

Introducción

La Ingeniería de Software surge el año 1968, con el objetivo de resolver los problemas de la crisis del software[1] a través de la incorporación de un conjunto de metodologías orientadas al desarrollo de productos software de calidad.
La ingeniería de software, es una disciplina que está en constante evolución, con mucha frecuencia surgen nuevas técnicas, metodologías y herramientas cuyo objetivo es mejorar la calidad y eficiencia de los productos de software, uno de los aspectos relevantes en este campo, es el que se refiere a los paradigmas para de construcción de software; cada paradigma está compuesto por una serie de métodos, herramientas y procedimientos, donde cada uno de ellos tiene ventajas y desventajas además de un campo de aplicación.
En la dinámica de esta evolución se introdujeron conceptos tales como:

- ✓ procedimientos y funciones, módulos, bloques estructurados, tipos de datos abstractos, cohesión, acoplamiento, patrón, herencia, etcétera. Estos conceptos proveen formas de abstracción de los problemas a resolver y constituyen el medio para lograr una programación de alto nivel; los productos software más importante se han desarrollado aplicando estos conceptos junto con tres principios estrechamente interrelacionados, tales como: abstracción, encapsulamiento, y modularidad.

La Programación Orientada a Aspectos o POA (en inglés: *aspect-oriented programming*) es un paradigma de programación reciente, que tiene un antecedente de aproximadamente 10 años; surge como una nueva forma de descomponer los sistemas, la intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de incumbencias[2] definiendo los aspectos de la aplicación.

En este trabajo se estudia la usabilidad de la POA a partir de una caracterización sucinta de las ventajas y desventajas que ofrece respecto a otras metodologías, propone el estudio y análisis de éste nuevo paradigma de programación; desde la definición de lo que es un aspecto hasta el estudio de su incursión dentro de los contenidos curriculares de la Carrera de Informática.

Métodos

Para alcanzar los objetivos del proyecto, establecer la usabilidad de la programación orientada a aspectos, se ha visto por conveniente realizar una investigación exploratoria del asunto a fin de tener una idea clara de este nuevo paradigma. La idea general era establecer el conjunto de propiedades que presenta el paradigma y obtener (inducir) una idea acerca de las posibilidades de su consideración en la formación de recursos humanos en la

[1] Es un término informático acuñado en 1968, en la primera conferencia organizada por la OTAN sobre desarrollo de software, de la cual nació formalmente la rama de la ingeniería del software. El término se adjudica a F.L.Bauer, aunque previamente había sido utilizado por Edsger dijkstra, en su obra The Humble Programmer. Básicamente, la crisis del software se refiere a la dificultad en escribir programas libres de defectos, fácilmente comprensibles, que terminen en el plazo y con los costos previstos, y que sean verificables. Las causas son, entre otras, la complejidad que supone la tarea de programar, y los cambios a los que se tiene que ver sometido un programa para ser continuamente adaptado a las necesidades de los usuarios.

[2] El término "separación de incumbencias" fue introducido en la década de 1970, por Edsger W. Dijkstra. Significa, simplemente, que la resolución de un problema dado involucra varios aspectos o incumbencias, los que deben ser identificadas y analizados en forma independiente.

Carrera de Informática. Desde ese punto de vista, los resultados que se buscan en el proyecto principalmente apuntan a:

- Dar una visión general, de tipo aproximativo, respecto al paradigma.
- Aumentar el grado de familiaridad con un paradigma desconocido.
- Obtener información sobre la posibilidad de llevar a cabo su inclusión en la formación de recursos humanos del área.
- Investigar problemas cruciales para su aplicación. Separación de incumbencias asociadas a los productos sw.
- Identificar conceptos o variables utilizadas. Tangled Code, Scattered Code, weaving, otros.
- Establecer prioridades y sugerir afirmaciones (postulados) verificables.
- Determinar tendencias e identificar relaciones potenciales entre variables y establecer una visión de utilidad del paradigma.

La investigación exploratoria: tiene por objeto esencial, familiarizarnos con un tema desconocido, novedoso o escasamente estudiado. Son el punto de partida para estudios posteriores de mayor profundidad. Sirve como punto de partida para familiarizarse con fenómenos desconocidos. Este tipo de estudio tiene mayor usabilidad cuando se necesita realizar un diagnóstico del proyecto.

Los estudios exploratorios en pocas ocasiones constituyen un fin en sí mismos, "por lo general determinan tendencias, identifican relaciones potenciales entre variables y establecen el `tono' de investigaciones posteriores más rigurosas". Se caracterizan por ser más flexibles en su metodología en comparación con los estudios descriptivos o explicativos, y son más amplios y dispersos que estos otros dos tipos (v.g., buscan observar tantas manifestaciones del fenómeno estudiado como sea posible). Asimismo, implican un mayor "riesgo" y requieren gran paciencia, serenidad y receptividad por parte del investigador (http://datateca.unad.edu.co/contenidos/204011/).

Resultados

Los productos son:

Producto 1: Identificación de las bases en las que se sustenta este nuevo paradigma. En el desarrollo de productos software además del diseño y la implementación de la funcionalidad básica (funciones relacionadas con las reglas de negocio) se hace necesario considerar funciones especiales orientadas a coadyuvar una explotación segura y eficiente del producto. La implementación de dichas funcionalidades especiales implica - en muchos casos- escribir líneas de código que resultan distribuidas por toda o gran parte de los módulos de la aplicación dando como resultado un código entremezclado (Tangled Code), lo que a futuro hace que tareas como el mantenimiento y/o el desarrollo de mejoras en la aplicación sean costosas.

En otras palabras, la incorporación de funcionalidades especiales en un sistema, tales como la sincronización, la distribución, la optimización de memoria, la gestión de seguridad, etcétera supone repetir líneas de código en diferentes componentes del sistema o bien incorporarlas en componentes con funcionalidades dispares. Con las técnicas de la programación procedural o la orientada a objetos no es posible resolver adecuadamente este problema ("código entrelazado *o tangled code").*

Según algunos autores, el mecanismo de composición -llamadas a procedimientos- que proveen los lenguajes de programación actuales, es el que da a lugar a que el código se entremezcle; pues a fin de implementar todas las funcionalidades de la aplicación –las del negocio y las complementarias- se procede a combinar de manera artificiosa muchas de las funciones dando como resultado el *tangled code.*

La programación orientada a aspectos es una alternativa que hace posible resolver el problema del código entremezclado a través de la siguiente consideración: una propiedad será implementada como un componente, si puede ser encapsulada de forma clara en un procedimiento y será implementada como un aspecto en caso contrario.

Los aspectos no resultan de descomponer el sistema, sino de identificar las necesidades que afectan al rendimiento de la aplicación o el comportamiento de los componentes en términos de las necesidades de cada aplicación. En consideración a lo anterior podemos decir que la programación orientada a aspecto tiene como objetivo: proveer al programador una técnica que le permita separar de manera lógica y simple los componentes –funciones del negocio- y los aspectos –funciones complementarias- asociados a una aplicación, evitando código entrecruzado (*scattering code*) y obteniendo de esta manera un producto fácil de mantener y/o mejorar.

La estructura de una aplicación implementada bajo la perspectiva POA es análoga a la estructura de una que se base en un lenguaje procedural; es decir, en el lenguaje procedural se requiere: i) un lenguaje, ii) un compilador para dicho lenguaje y iii) un programa escrito en el anterior lenguaje; por otro lado, una aplicación orientada a aspectos requiere de: 1.1. un lenguaje de componentes -el utilizado para programar los componentes; 1.2 uno o más lenguajes de aspectos -para programar los aspectos-; 2.1) una herramienta para combinar los aspectos y los componentes (*aspect weaver*); 3.1) un programa que implemente los componentes usando el lenguaje de componentes; 3.2) uno o más programas que implementen los aspectos usando los lenguajes de aspectos.
El proceso de entrelazado de código dependiendo del lenguaje y la herramienta de entrelazado, puede hacerse en tiempo de ejecución (*run-time weaving*) o en tiempo de compilación (*compile-time weaving*).
POA soporta la descomposición orientada a objetos, la descomposición procedimental y la descomposición funcional; se afirma que el estado de investigación de este nuevo paradigma está en proceso de desarrollo; en este sentido, la programación orientada a aspectos constituye la quinta generación de métodos de desarrollo de sistemas.

Producto 2. Definición de la estructura del POA. El POA está definido sobre la base de tres requerimientos principales:

a) Un lenguaje para definir la funcionalidad básica, conocido como lenguaje base o componente. Podría ser un lenguaje imperativo, o un lenguaje no imperativo (C++, Java, Lisp, ML).

b) Uno o varios lenguajes de aspectos, para especificar el comportamiento de los aspectos. (COOL, para sincronización, RIDL, para distribución, AspectJ, de propósito general.)

c) Un tejedor de aspectos (Weaver), que se encargará de combinar los lenguajes. Proceso que puede hacerse en tiempo

de ejecución o en tiempo de compilación.

Gráficamente podríamos sintetizarlo de la siguiente manera:

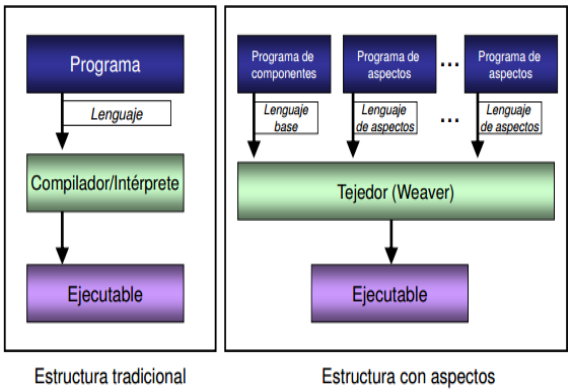

Figura 1: Estructura de la POA (Asteasuain, 2002)

Producto 3. Definir las fases del POA**.** Las fases del paradigma POA están en desarrollo o en etapa de experimentación por lo que podríamos mencionar que estas deben considerar las siguientes pautas:

- Los aspectos dinámicos de la aplicación deben separarse claramente de los aspectos estáticos; como por ejemplo, a través de un constructor o palabra reservada que indique la naturaleza del aspecto.
- La especificación de la naturaleza del aspecto debe ser opcional.
- El alcance de la influencia de dicha especificación tiene que ser controlado de una forma natural y con suficiente granularidad. Esta pauta ayuda al programador a entender y razonar sobre su programa.
- El conocimiento que tiene el cliente sobre los detalles internos de implementación debe ser mínima. Por ejemplo, el cliente podría elegir diferentes niveles de seguridad para el aspecto Seguridad-Edificio (alto, mediano o bajo) en ejecución, sin conocer cómo está implementada cada alternativa.

Producto 4. Definir herramientas del POA. El desarrollo de Software es un proceso complejo y a menudo difícil que requiere la síntesis de muchos elementos, a continuación se mencionan los más importantes:

- Modelar y analizar los procesos de negocios
- Construir diseño y modelos de comportamientos
- Generar e importar código fuente en una variedad de lenguajes
- Generar e importar esquema de base de datos
- Generar e importar XSD
- Crear modelos de componentes y de despliegue
- Rastrear cambios
- Administrar pruebas
- Confirmar la trazabilidad desde los requisitos a través y hasta el despliegue
- Documentar su desarrollo de software
- Comunicar y desarrollar proyectos de ingeniería de software basados en el equipo
- Modelado/ingeniería rápida de su desarrollo de software

Existen diversos tipos de herramientas con fines muy diversos, a continuación se menciona algunas:

- Sistema de seguimiento de errores: *Bugzilla*
- Generador de código: *Make*
- Conversor de código: *JTest*
- Compilador: gcc
- Depurador: gdb
- Desensamblador: *OllyDbg*
- Enlazador: ---
- Generador de documentación: Javadoc
- Generador de GUI: *NetBeans* cuenta con uno.

- Detector de fugas de memoria: *dmalloc*
- Analizador sintáctico: *Lex*
- Profilers: Java Virtual Machine Tools Interface (JVM TI)
- Control de versiones: CVS
- Herramienta de búsqueda: *grep*
- Entorno de desarrollo integrado: *Netbeans*
- Generador de estilo: *indent*
- Editor de texto: *jEdit*

Durante el diseño detallado se confirma la corrección de la arquitectura y que se cumplen los requisitos. Las decisiones que toma el diseñador durante el Diseño OA (DOA) se basan en las decisiones tomadas durante la ingeniería de requisitos o el diseño arquitectónico; de esta manera, cuando los requisitos o la arquitectura cambian, las decisiones tomadas tienen que revisarse y posiblemente haya que alterarlas. Por tanto, el objetivo del DOA, como cualquier actividad de diseño del software, es caracterizar y especificar el comportamiento y la estructura de los sistemas.

Los modelos de DOA se pueden considerar ligados a extensiones de UML o independientes de este lenguaje de modelado:

i) Extensiones de UML orientadas a aspectos: Los trabajos desarrollados en este ámbito representan los conceptos de orientación a aspectos a nivel de diseño; este tipo de modelos se incluye dentro del llamado Modelado Orientado a Aspectos o *Aspect Oriented Modelin*g. Una de estas extensiones es la propuesta de Stein: Modelado de Diseño Orientado a Aspectos o *Aspect-Oriented Design Modeling* (AODM) que extiende UML con una notación de diseño para soportar conceptos OA; estos elementos están próximos a *AspectJ*. El modelo soporta la especificación del comportamiento y la estructura transversal del sistema; ésta se expresa en los diagramas de clases y con una plantilla parametrizada de diagramas de colaboración.

ii) Automatización de la transformación desde modelos de diseño de alto nivel a plataformas de implementación específicas. *CoCompose*, es un lenguaje de diseño no orientado a UML, soporta la representación de aspectos reutilizables de alto nivel como *"feature*": una feature es un aspecto de alto nivel que atraviesa los límites de las aplicaciones. Las *features* son también construcciones abstractas que describen patrones de diseño, estando por tanto semánticamente bien definidos. *CoCompose* es un lenguaje de diseño gráfico que se puede usar para realizar DOA; también soporta diseños ejecutables.

- Programación Orientada a Aspectos. Para la implementación de las aplicaciones OA es recomendable utilizar lenguajes que soporten el concepto de aspecto; los Lenguajes de Programación Orientados a Aspecto (LPOA). El modelo de diseño adoptado en este paradigma determinará la conveniencia de utilizar un LPOA u otro. Hay varios lenguajes de programación OA:
  - *AspectJ*, que es una extensión de Java, es el más importante y el más popular,
  - *CaesarJ* es un modelo de programación POA con su propio lenguaje de programación; es un modelo dependiente de la tecnología, se caracteriza por desarrollar módulos de alto nivel para expresar los aspectos

independientemente de los puntos de unión (join points),

- *DemeterJ* fue concebido inicialmente como un modelo de componentes; su evolución llevó hacia la definición de componentes aspectuales (*aspectual components*) en los cuales, con una mínima modificación, se pueden especificar aspectos,
- JAC y JBOSS AOP están relacionados con *AspectJ*.

La POA es una de las ramas con mayor futuro dentro de la ingeniería de software; el estudio de esta una nueva tecnología nos permitirá en un futuro cercano ampliar los recursos para favorecer la calidad del desarrollo de software. Se ha descrito las características, definiciones y elementos fundamentales de la arquitectura del software y se ha revisado los lenguajes de descripción arquitectónica que se han considerado más interesantes.

Se ha mostrado importancia de la AS en el desarrollo de los sistemas software; en este sentido, se han propuesto mecanismos para describir formalmente las arquitecturas, en concreto, mediante la definición de los diferentes LDA; se ha mostrado la importancia de la AS durante la evolución y el interés de considerar las AS dinámicas en el desarrollo de sistemas complejos.

Producto 5. Definición de las ventajas y desventajas del POA. En una primera impresión, la programación orientada a aspectos y la programación orientada a objetos pareciera que son en realidad el mismo paradigma, no obstante, esta noción es errónea. En la programación orientada a objetos los sistemas se modelan como un conjunto de objetos que interactúan entre sí, sin embargo, falla al modelar los conceptos que se entrecruzan. La diferencia radica en que mientras la programación orientada a aspectos se enfoca en los conceptos que se entrecruzan, la programación orientada a objetos se enfoca en los conceptos comunes.

De la consecución de estos objetivos se pueden obtener las siguientes ventajas:

- Un código menos enmarañado, más natural y más reducido.
- Una mayor facilidad para razonar sobre las materias, ya que están separadas y tienen una dependencia mínima.
- Más facilidad para depurar y hacer modificaciones en el código.
- Se consigue que un conjunto grande de modificaciones en la definición de una materia tenga un impacto mínimo en las otras.
- Se tiene un código más reusable y que se puede acoplar y desacoplar cuando sea necesario.

Producto 6. Definición de las ventajas y desventajas de otros paradigmas. A medida que la ingeniería de software fue creciendo, se fueron introduciendo conceptos que llevaron a una programación de más alto nivel: la noción de tipos, bloques estructurados, agrupamientos de instrucciones a través de procedimientos y funciones como una forma primitiva de abstracción, unidades, módulos, tipos de datos abstractos, genericidad, herencia. Como vemos, los progresos más importantes se han obtenido aplicando tres principios, los cuales están estrechamente relacionados entre sí: abstracción, encapsulamiento, y modularidad.

Un avance importante lo introdujo la Programación Orientada a Objetos (POO), donde se fuerza el encapsulamiento y la abstracción, a través de una unidad que captura tanto funcionalidad como comportamiento y estructura interna. A esta entidad se la conoce como clase. La clase hace énfasis tanto en los algoritmos

como en los datos. La POO está basada en cuatro conceptos:

- Definición de tipos de datos abstractos.
- Herencia
- Encapsulamiento.
- Polimorfismo por inclusión.

La herencia es un mecanismo lingüístico que permite definir un tipo de dato abstracto derivándolo de un tipo de dato abstracto existente. El nuevo tipo definido "hereda" las propiedades del tipo padre.

Ya sea a través de la POO o con otras técnicas de abstracción de alto nivel, se logra un diseño y una implementación que satisface la funcionalidad básica, y con una calidad aceptable. Sin embargo, existen conceptos que no pueden encapsularse dentro de una unidad funcional, debido a que atraviesan todo el sistema, o varias partes de él (*crosscutting concerns*). Algunos de estos conceptos son: sincronización, manejo de memoria, distribución, chequeo de errores, *profiling*, seguridad o redes, entre otros.

Las técnicas de implementación actuales tienden a implementar los requerimientos usando metodologías de una sola dimensión, forzando los requerimientos a ser expresados en esa única dimensión. Esta dimensión resulta adecuada para la funcionalidad básica y no para los requerimientos restantes, los cuales quedan diseminados a lo largo de la dimensión dominante. Es decir, que mientras el espacio de requerimientos es de n-dimensiones, el espacio de la implementación es de una sola dimensión. Dicha diferencia resulta en un mapeo deficiente de los requerimientos a sus respectivas implementaciones. Algunos síntomas de este problema pueden ser categorizados de la siguiente manera:

1. Código Mezclado (Code Tangling): En un mismo módulo de un sistema de software pueden simultáneamente convivir más de un requerimiento. Esta múltiple existencia de requerimientos lleva a la presencia conjunta de elementos de implementación de más de un requerimiento, resultando en un Código Mezclado.

2. Código Diseminado (Code Scattering): Como los requerimientos están esparcidos sobre varios módulos, la implementación resultante también queda diseminada sobre esos módulos. Estos síntomas combinados afectan tanto el diseño como el desarrollo de software, de diversas maneras:

- Baja correspondencia: la implementación simultánea de varios conceptos oscurece la correspondencia entre un concepto y su implementación, resultando en un pobre mapeo.
- Menor productividad: la implementación simultánea de múltiples conceptos distrae al desarrollador del concepto principal, por concentrarse también en los conceptos periféricos, disminuyendo la productividad.
- Menor rehúso: al tener en un mismo módulo implementado varios conceptos, resulta en un código poco reusable.
- Baja calidad de código: el Código Mezclado produce un código propenso a errores. Además, al tener como objetivo demasiados conceptos al mismo tiempo se corre el riesgo de que algunos de ellos sean subestimados.
- Evolución más dificultosa: como la implementación no está completamente modularizada los futuros cambios en un requerimiento implican revisar y modificar cada uno de los módulos

donde esté presente ese requerimiento. Esta tarea se vuelve compleja debido a la insuficiente modularización.

Tenemos entonces que las descomposiciones actuales no soportan una completa separación de conceptos, la cual es clave para manejar un software entendible y evolucionable. Podemos afirmar entonces que las técnicas tradicionales no soportan de una manera adecuada la separación de las propiedades de aspectos distintos a la funcionalidad básica, y que esta situación tiene un impacto negativo en la calidad del software.

Como se mencionó anteriormente, la (POA) nace como respuesta a este problema. La POA es un desarrollo que sigue a la POO, y como tal, soporta la descomposición orientada a objetos, además de la procedimental y la funcional. Sin embargo, la programación orientada a aspectos no es una extensión de la POO, ya que puede utilizarse con los diferentes estilos de programación mencionados anteriormente.

Teniendo en cuenta nuevamente la forma en que la ingeniería de software ha crecido, siempre de la mano de nuevas formas de descomposición que implicaron luego nuevas generaciones de sistemas, es válido preguntarnos si también de la mano de la POA nacerá una nueva generación de sistemas de software.

Discusión

En el desarrollo de sistemas de Software, además del diseño y la implementación de la funcionalidad básica, está presentes otros aspectos tales como: la sincronización, la distribución, el manejo de errores, la optimización de la memoria, la gestión de seguridad, etc., en el desarrollo del código, merece prestar mucha atención a dichas tareas en cada una de las funcionalidades del sistema; esta alternativa aunque viable es muy restringida pues si el código del sistema cambia o se introducen otras funcionalidades será necesario revisar el sistema completo a fin de evitar problemas que podrán ir en contra de la fiabilidad del sistema.

La programación orientada a aspectos constituye uno de los avances más importantes de la Ingeniería de software en los últimos años, es una nueva metodología de diseño que aspira resolver los problemas de los métodos tradicionales; es decir intenta resolver de manera simple aspectos relacionados a las funcionalidades básicas de los sistemas y aspectos principalmente los complementarios que deberían presentar cada una de las funcionalidades del sistema software mencionados anteriormente. Lo que se pretende es implementar una aplicación de forma eficiente, inteligible y fácil de mantener.

Desde el punto de vista teórico, este paradigma descompone los sistemas en términos de datos necesarios en las diferentes funcionalidades del sistema; además de las relaciones de cada una de estas.

Conclusiones

La programación orientada a objetos (POO) supuso un gran paso en la ingeniería del software, ya que presentaba un modelo de objetos que parecía encajar de manera adecuada con los problemas reales. La cuestión era saber descomponer de la mejor manera el dominio del problema al que nos enfrentáramos, encapsulando cada concepto en lo que se dio en llamar objetos y haciéndoles interactuar entre ellos, habiéndoles dotado de una serie de propiedades. Surgieron así numerosas metodologías para ayudar en tal proceso de

descomposición y aparecieron herramientas que incluso automatizaban parte del proceso. Esto no ha cambiado y se sigue haciendo en el proceso de desarrollo del software. Sin embargo, frecuentemente la relación entre la complejidad de la solución y el problema resuelto hace pensar en la necesidad de un nuevo cambio. Así pues, nos encontramos con muchos problemas donde la POO no es suficiente para capturar de una manera clara todas las propiedades y comportamientos de los que queremos dotar a nuestra aplicación. Así mismo, la programación procedural tampoco nos soluciona el problema.

Entre los objetivos que se ha propuesto la POA están, principalmente, el de separar conceptos y el de minimizar las dependencias entre ellos. Con el primer objetivo se persigue que cada decisión se tome en un lugar concreto y no diseminada por la aplicación. Con el segundo, se pretende desacoplar los distintos elementos que intervienen en un programa. Su consecución implicaría las siguientes ventajas:

- Un código menos enmarañado, más natural y más reducido.
- Mayor facilidad para razonar sobre los conceptos, ya que están separados y las dependencias entre ellos son mínimas.
- Un código más fácil de depurar y más fácil de mantener.
- Se consigue que un conjunto grande de modificaciones en la definición de una materia tenga un impacto mínimo en las otras.
- Se tiene un código más reusable y que se puede acoplar y desacoplar cuando sea necesario.

Se ha establecido que el Paradigma Orientado a Aspectos aplica el enfoque sistémico en el proceso de desarrollo de productos software, pues introduce en el análisis tareas relacionadas con el diseño, la implementación y el mantenimiento del producto, buscando la implementación de un código limpio y fácilmente mantenible.

Dadas estas características el Paradigma Orientado a Aspectos daría la posibilidad de desarrollarlas con mayor calidad y posibilitaría la actualización de productos software desarrollados bajo otros paradigmas.

- Ejemplo de aplicación de la poa. De acuerdo a lo desarrollado en el trabajo presente, la POA busca resolver problemas de reuso de código; trata de la separación de aspectos tratando de modularizar los asuntos transversales (modularización de incumbencias), donde la POO no llega a dar soluciones definitivas.

  Las técnicas que generalmente se usan en el diseño de un sistema se basan en el principio de “divide y vencerás”, en el que un sistema es descompuesto en varios subsistemas, donde de manera implícita o explícita está la visión jerárquica del sistema. La desventaja de estas técnicas, es que en esta descomposición, pueden existir aspectos que se vayan repitiendo en los diferentes subsistemas.

Ejemplo 1: Un ejemplo sencillo que describe el comportamiento de la POA se refiere a: un domicilio particular pueden existir en una misma habitación un televisor, un reproductor de video y un equipo de música, cada uno con su propio control y cada uno trabaja perfectamente, se trata de solucionar el problema de la existencia de muchos controles encontrando un único control universal que pueda manejar todos los equipos (televisor, reproductor, equipo de música).

Ejemplo 2: (Extraído de Moreno, 2015. Acápites 4 y 5). Un procesador de imágenes. Este ejemplo está sacado del trabajo publicado en 1997 para la *European Conference on Object Oriented Programming* (ECOOP) celebrada en Finlandia. Supongamos una aplicación de procesamiento de imágenes en blanco y negro. El dominio del problema son las imágenes que deben pasar por una serie de filtros para obtener el resultado deseado. Se podrían hacer tres implementaciones: una que fuese fácil de entender pero ineficiente, una eficiente pero difícil de entender y una tercera, basada en POA, que a la vez sería fácil de entender y eficiente. Para mayor y completa información, se puede referir a la siguiente dirección: http://www.geocities.ws/nulain/computing/Intro_a_la_POA.pdf

Referencias

Amparo Navasa, M. A. (s.f.). AspectLEDA: *Un Lenguaje de Descripción Arquiectónica Orientado a Aspectos*. España: Departamento de Informática. Universidad de Extremadura.

Andrés, J. O. (s.f.). *El Proceso del Desarrollo de Software*. Obtenido de El Proceso del Desarrollo de Software: http://es.slideshare.net/laos7/el-proceso-de-desarrollo-de-software

Arregui, J. J. (2005). *Revisión Sistemática de Métricas de Diseño Orientado a Objetos*. Madrid, España: Universidad Politécnica de Madrid, Facultad de Informática.

Asteasuain, B. E. (2002). *Programación Orientada a Aspectos*. Bahía Blanca, Argentina: Universidad Nacional del Sur.

Fernando Asteasuain, B. C. (2003). *Programación Orienada a Aspectos: Metricas y Evaluación*. Obtenido de http://www.academia.edu/2839498/Programaci%C3%B3n_Orientada_a_Aspectos_Metodolog%C4%B1a_y_Evaluaci%C3%B3n

Gregor Kiczales, J. L. (1997). *Aspect Oriented Programming*. Finland: Springer - Verlag.

Historia de los paradigmas en el desarrollo de software. (2007). Obtenido de http://sistemas8ittg.obolog.es/etiqueta_historia-de-los-paradigmas-en-el-desarrollo-del-so

http://sedici.unlp.edu.ar/bitstream/handle/10915/4057/2_-_Ingenier%C3%ADa_de_requerimientos.pdf?sequence=4.

(s.f.). *Ingeniería de Requerimientos*. Obtenido de http://sedici.unlp.edu.ar/bitstream/handle/10915/4057/2_-_Ingenier%C3%ADa_de_requerimientos.pdf?sequence=4

Ingeniería del Software. (2010). Obtenido de http://histinf.blogs.upv.es/2010/12/28/ingenieria-del-software/

Martinez, A. N. (2008). *Marco de trabajo para el desarrollo de arquitecturas software orientado a aspectos*. Cáceres, España: Universidad de extremadura.

Meyer, B. (1988). Object- oriented software construction. Prentice Hall.

Moreno, J. M. (2015). *Introducción a la Programación Orientada a Aspectos.* Obtenido de Introducción a la Programación Orientada a Aspectos: http://www.geocities.ws/nulain/computing/Intro_a_la_POA.pdf

Paradigmas de la Ingeniería de Software. (2015). Obtenido de

http://www.sites.upiicsa.ipn.mx/polilibros/portal/Polilibros/P_proceso/ANALISIS_Y_DISEnO_DE_SISTEMAS/IngenieriaDeSoftware/CIS/UNIDAD%20I/1.5.htm
Pressman,

R. S. (2002). Ingenieria del Software . En R. S. Pressman, *Ingenieria del Software* (pág. 579). Madrid: McGraw Hill.

Quintero, A. M. (2000). *Visión General de la Programación Orientada a Aspectos.* España: Departamento de Lenguajes Informáticos, Universidad de Sevilla.

Standrads, E. (2004). *Introducción a la Programacion Orientada a Aspectos*. Obtenido de http://programacion.net/articulo/introduccion_a_la_aop_programacion_orientada_al_aspecto_260#2_marcos

Thomas, P. J. (2005). Definición de un proceso de Elicitación de requerimientos. La Plata, Argentina.

Universidad del Valle - Escuela de Sistemas y Computación. (2008). *Requerimientos de software*. Obtenido de http://eisc.univalle.edu.co/materias/Material_Desarrollo_Software/2008/DS2-sesion4-Requerimientos.pdf

Valdiviezo, E. Y. (s.f.). *Programación Orientada a Aspectos.* Ecuador: Universidad Técnica particular de Loja.
Venegas, R. A. (2008). *Programación Qrientada a Aspectos*. Chile: Universidad del Bio-Bio.

Presentado: La Paz, 9 de octubre de 2015

Aceptado: La Paz, 27 de noviembre de 2015