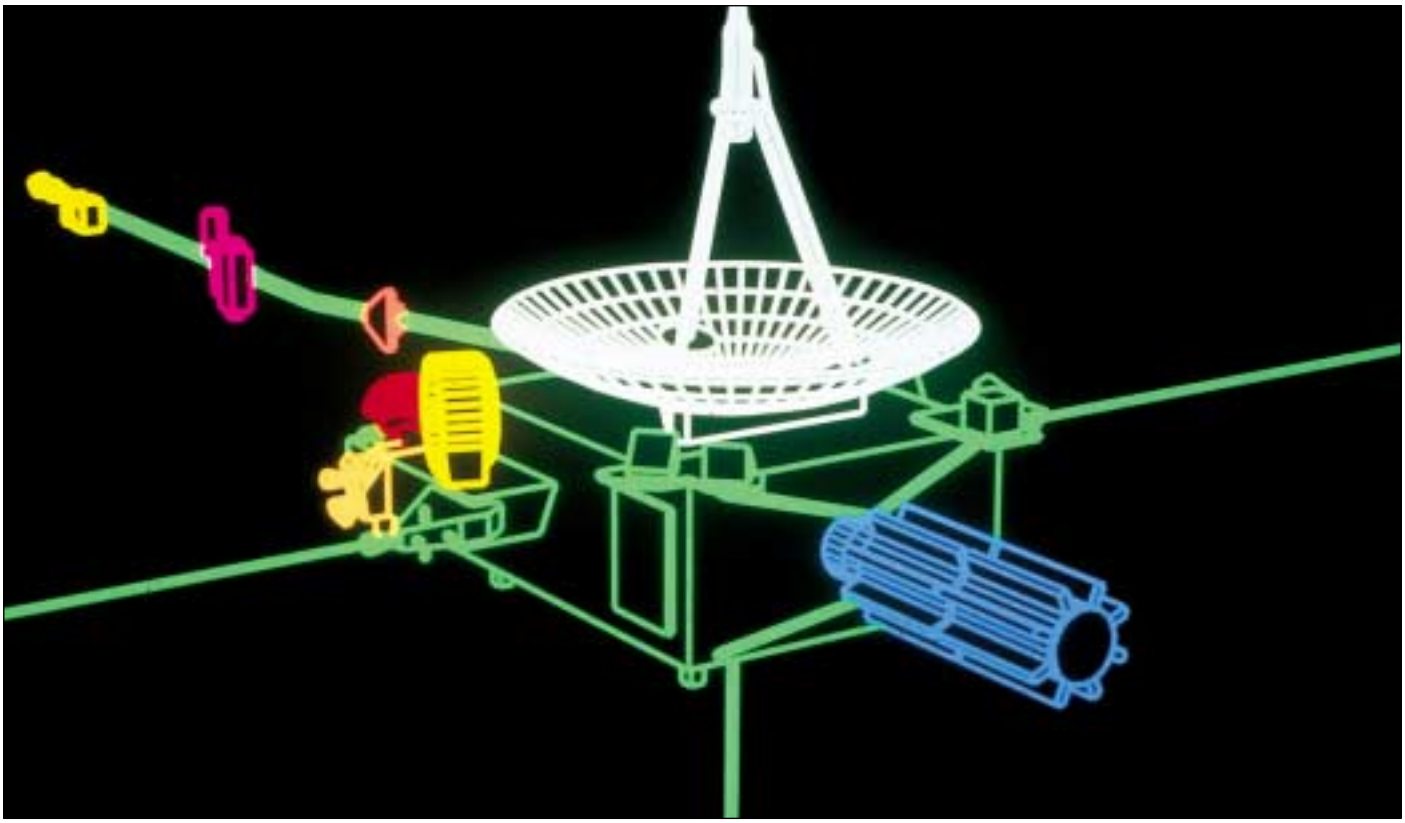
En el desarrollo de sistemas distribuidos existen numerosos problemas causados por el dinamismo y evolución de dichos sistemas. Ante esta situación, estos sistemas deben ser diseñados para poder ser adaptados fácilmente al entorno evolutivo que les rodea que hace que sus requisitos a satisfacer cambien continuamente. Pero esto no es nada fácil.
Por este motivo, ha surgido en los últimos años la programación orientada a aspectos (POA) como una magnífica alternativa para implementar adaptabilidad y reutilización de sistemas en general.

PaDA: Patrón de Distribución con Aspectos

María Francisca Rosique Contreras,
paquirosique@hotmail.com



Cuando implementamos un sistema distribuido, este requiere una alta modularidad, el sistema debe tener una distribución independiente de preocupaciones (termino ingles *concerns*). Para lograr una mejor separación de preocupaciones debemos usar programación orientada a aspectos. Un aspecto delimita una preocupación transversal, por ejemplo, distribución, se teje

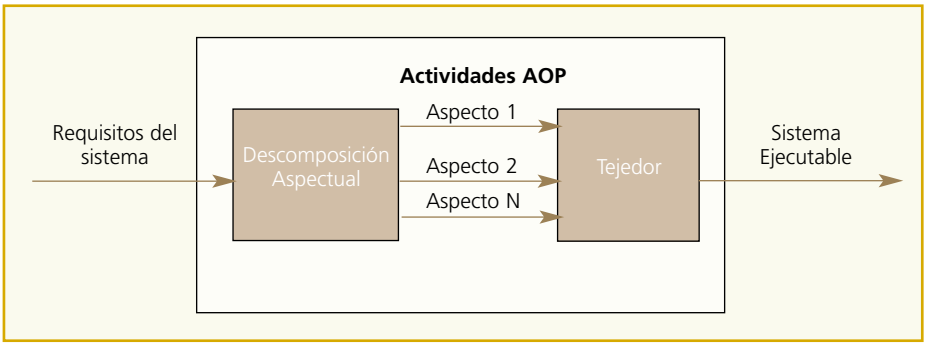


Figura 1: Fases de desarrollo de AOP

automáticamente a un sistema cambiando su conducta original. Por eso, se debe ejecutar el sistema con un lenguaje de programación que sea compatible con la programación orientada a aspectos, en nuestro caso con Java.

Código mezclado (código con preocupaciones diferentes entrelazados el uno con otro) y código esparcido (código con respecto a una preocupación distribuido en varias unidades del sistema) reducen la modularidad del sistema. Por eso, el mantenimiento y la extensibilidad se ven disminuidas también.

PaDA (Patrón para Distribución con Aspectos) proporciona una estructura para implementar un código distribuido mediante programación orientada a aspectos, logrando así una mejor separación de preocupaciones transversales (*crosscutting concern*). Aumentando la modularidad, extensibilidad y mantenimiento del sistema.

NECESIDADES

Para distribuir un sistema, PaDA tiene en cuenta una serie de necesidades, alguna de ellas propias de los sistemas distribuidos:

— Comunicación Remota. La comunicación entre dos componentes de un sistema debe ser remota para permitir que varios clientes accedan al sistema, considerando que la interfaz del usuario es la parte distribuida del sistema.

— API independiente. El sistema debe ser completamente independiente de la API de comunicaciones y el middleware, para facilitar el mantenimiento del sistema, no se mezcla el código con las comunicaciones o el código interfaz usuario. También permite cambiar la API de comunicación sin modificar otro código del sistema.

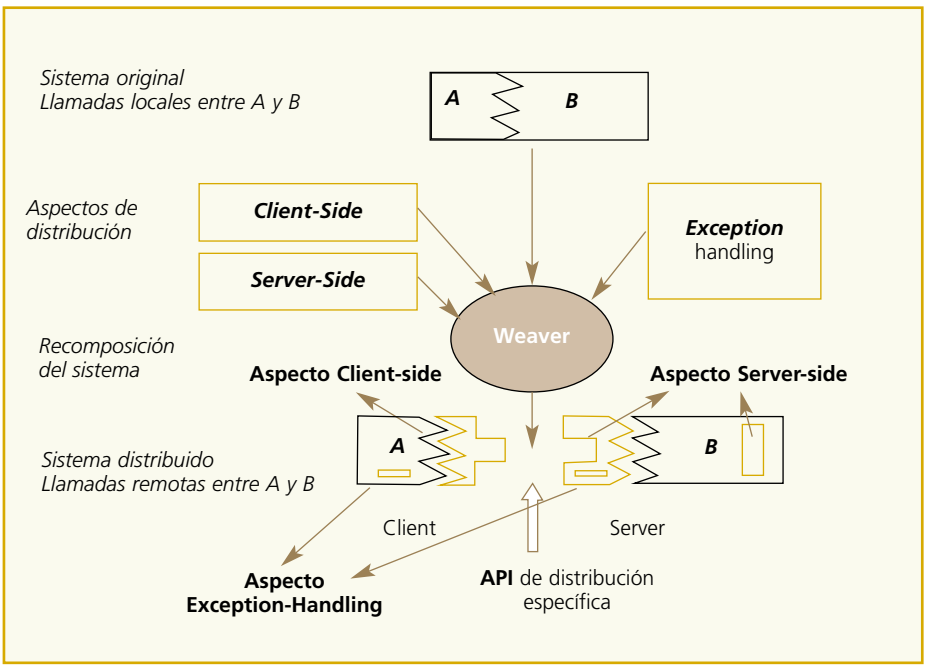


Figura 2: Estructura PaDA

— Un mismo sistema puede emplear middleware diferentes al mismo tiempo. Por ejemplo, dos clientes acceden el sistema, uno usa RMI y el otro CORBA.

— Cambio dinámico de middleware. El sistema debe permitir cambia el middleware sin cambiar o recompilar su código fuente.

— Facilita pruebas funcionales. La prueba funcional facilita la comprobación del sistema con su versión local; por eso, los errores del código de la distribución no afectarán a las pruebas.

Para resolver el problema previamente presentado, PaDA usa programación orientada a aspectos para definir aspectos de la distribución que se pueden tejer al código fuente del sistema central. Esta separación de preocupaciones se consigue definiendo aspectos que implementan preocupaciones específicas. Después los aspectos de un sistema, se pueden tejer automáticamente con el código fuente del sistema, creándose así una versión del sistema con los requerimientos del aspecto.

La Figura 1 ilustra la descomposición aspectual, que identifica las preocupaciones transversales (*crosscutting concerns*) implementando aspectos de un sistema, y la recomposición aspectual, o sistema tejido, que compone el aspecto que implementa las preocupaciones transversales identificadas, con el sistema, obteniendo la versión decisiva con las funciones requeridas. En nuestro caso PaDA define una sola preocupación (distribución), que es implementado con tres aspectos, como mostramos a continuación.

ESTRUCTURA DE PaDA

El patrón PaDA define tres aspectos: uno que atraviesa o corta el módulo servidor, otro que atraviesa o corta las clases del cliente, y un tercero que atraviesa o corta a ambos, que provee el manejo de excepciones, como se muestra en la Figura 2. De hecho, el tercer aspecto define una preocupación transversal que es la de distribución en si, es decir el manejo de la excepción. De hecho, el aspecto que atraviesa al módulo servidor también atraviesa o corta otras clases que son tipos de argumentos o tipos de retorno del método sobre el que actúa.

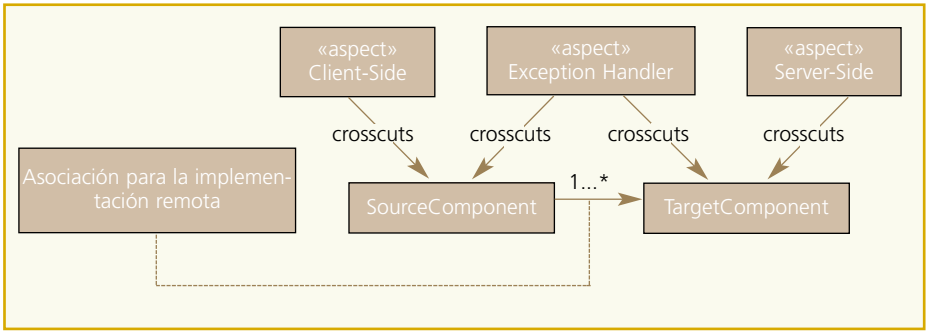


Figura 3: Diagrama UML de las clases PaDA

La Figura 3 representa un diagrama UML que muestra los aspectos y los componentes que permiten el acceso remoto.

DINÁMICAS DE PaDA

La Figura 4 muestra en un diagrama de secuencia cual es el comportamiento original del sistema l: una instancia de SourceComponent ejecuta una llamada local a unos métodos de una instancia de TargetComponent.

La Figura 5 muestra el diagrama de secuencias del comportamiento del sistema después de haber tejido los aspectos de distribución al sistema. Las llamadas locales de *SourceComponent* son interceptadas por el aspecto *ClientSide* que obtiene la referencia de la instancia remota y redirecciona la llamada local a dicha instancia. El aspecto *ServerSide* crea y ejecuta la instancia remota (una instancia de *TargetComponent*) disponible para las respuestas a llamadas remotas.

Si la llamada remota causa una excepción, como ocurre en la invocación al método n(), el aspecto *ExceptionHandler* trata la excepción como una excepción no chequeada (*unchecked*) y la lanza en el *ServerSide*. El mensaje que trata y lanza la excepción no chequeada es un mensaje dirigido al aspecto *ExceptionHandler,* porque el aspecto

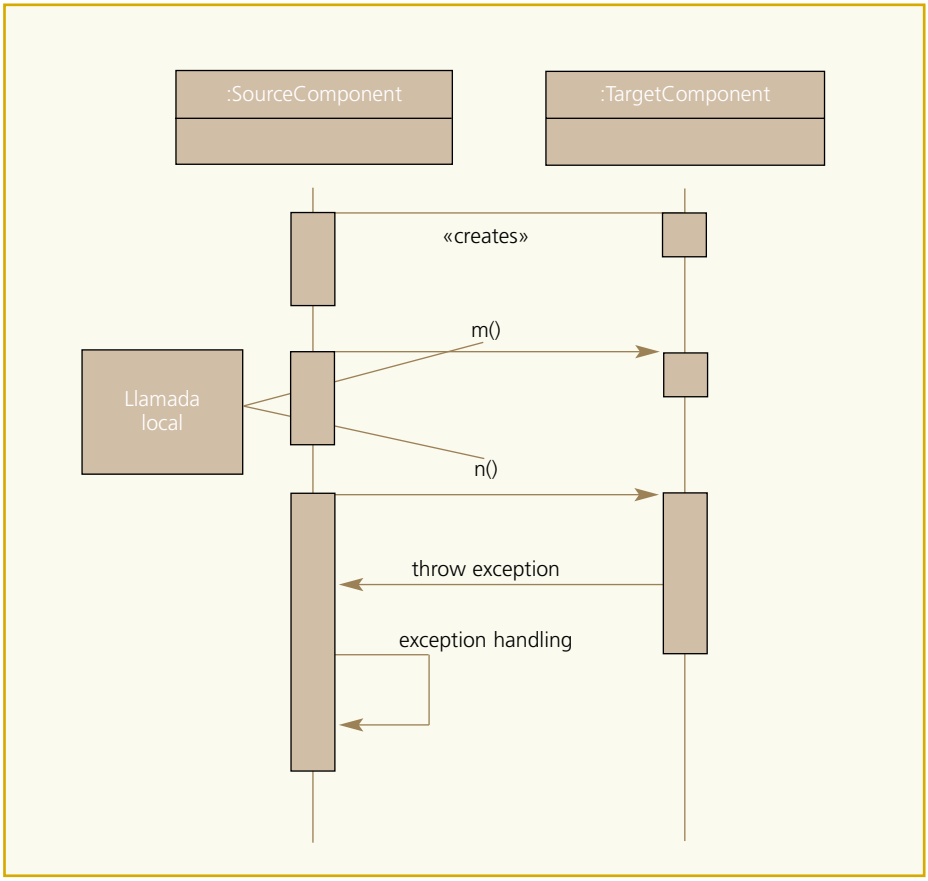


Figura 4: Comportamiento original del sistema

ExceptionHandler es responsable, por capturar la excepción no chequeada, de suministrar el manejo necesario de la excepción en el *ClientSide* (SourceComponent).

VENTAJAS E INCONVENIENTES

Ventajas

— *Implementación distribuida.* El patrón proporciona comunicación remota entre dos módulos de un sistema; podemos convertir un sistema cualquiera en distribuido simplemente aplicando PaDA.

— *Modularidad.* En la estructura PaDA el código de los aspectos de distribución está completamente separado del código del sistema, ejecutando el código fuente del sistema API independientemente.

— *Mantenimiento y extensibilidad.* Como el código de la distribución está completamente separado del código del sistema, cambiar la API comunicación es más simple y no causa ningún impacto en el código del sistema. Los programadores sólo deben escribir otros aspectos de la distribución, para el nuevo API especificado, o cambiar los aspectos ya implementados para corregir errores y tejerlos al código fuente del sistema original.

— *Separación adicional de preocupaciones.* La estructura PaDA define el manejo de excepciones como un nuevo aspecto a tratar que no se realiza por técnicas de la programación orientada a objetos. Por eso, se puede cambiar el manejador de la excepción sin modificar el código fuente del sistema original y en los aspectos de la distribución, o e n los otros aspectos que implementan los requerimientos del sistema.

— *Facilita comprobación de requisitos funcionales.* Se pueden hacer pruebas de los requisitos funcionales fácilmente si ejecutamos el sistema sin la distribución. La completa separación de aspectos preserva el código fuente del sistema original. Esto significa que se añaden los aspectos de distribución al sistema sólo si se realiza la composición aspectual (*weaving*).

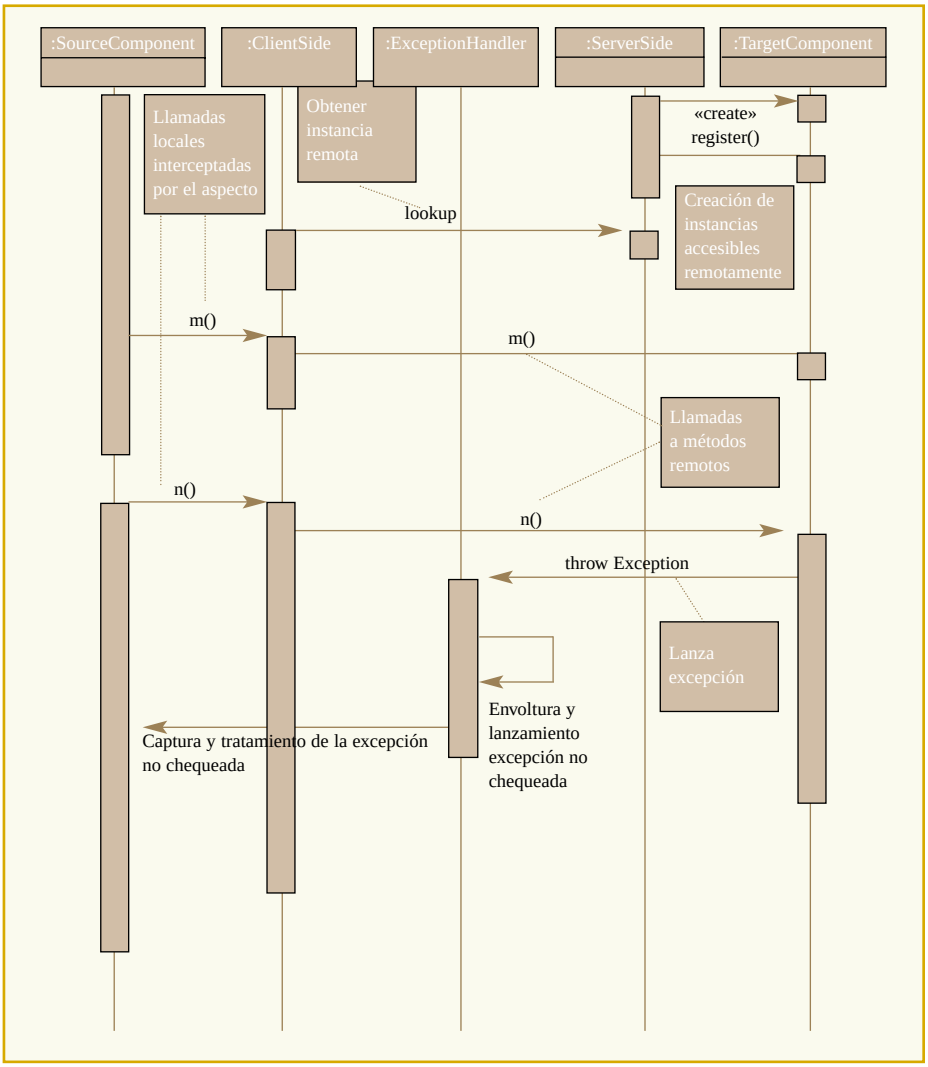


Figura 5: Comportamiento del sistema después de aplicarle PaDA

Inconvenientes

El patrón PaDA también tiene una serie de inconvenientes:

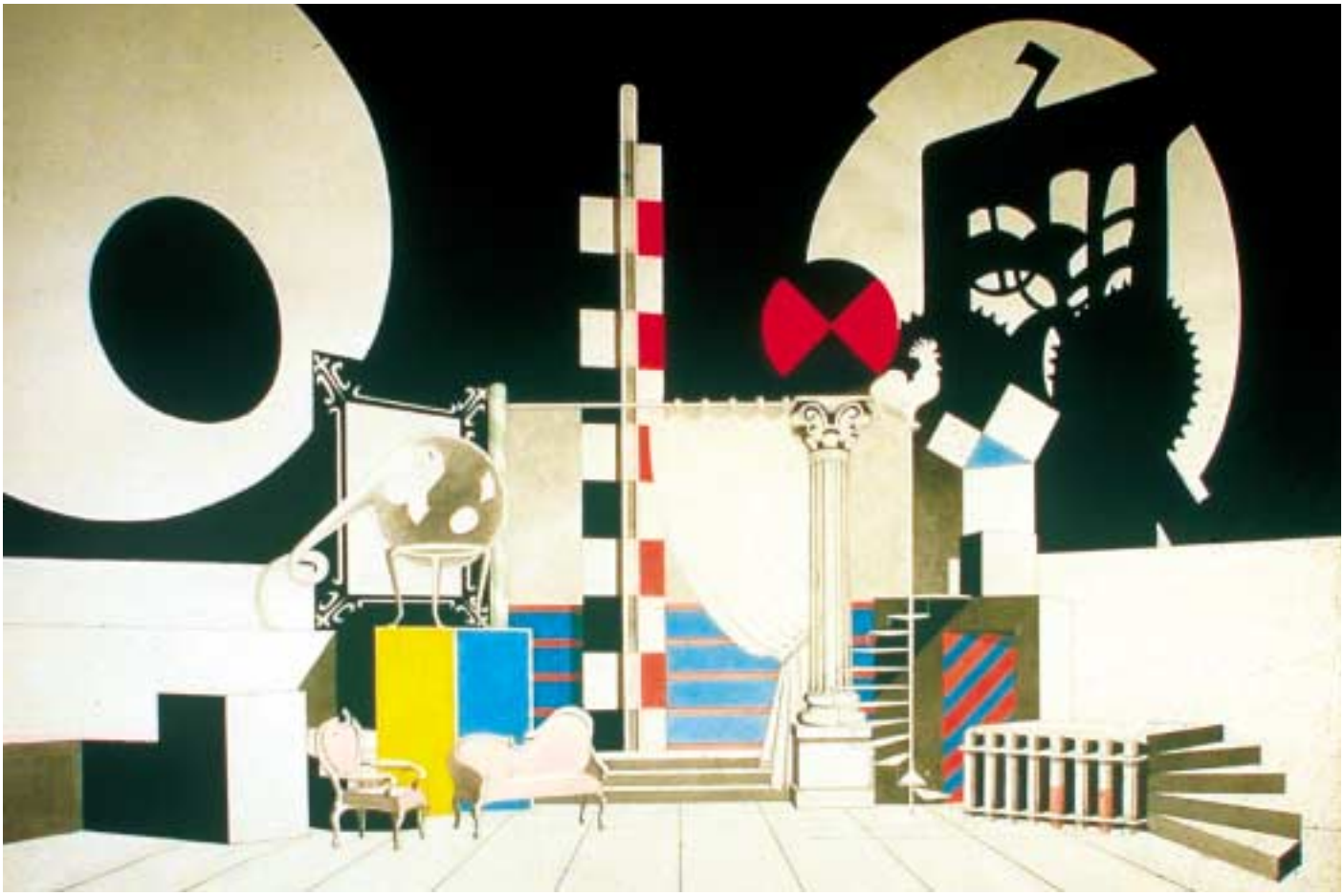
— *Nuevo paradigma de la programación.* El patrón usa una nueva técnica de la programación que implica saber un nuevo paradigma de la programación para usar el patrón. Otro cambio de tener un nuevo paradigma de la programación es con respecto a la separación del código que usualmente estaba junto en el mismo módulo. El programador de los requisitos funcionales no puede ver el código resultante que implementara el aspecto requerido, decreciendo la legibilidad del código y desconfianza del usuario.

— *Aumento del número de módulos.* PaDA añade tres módulos nuevos en el sistema, aumentando la complejidad del manejo de los módulos.

— *Aumento del bytecode.* Cada aspecto definido dará por resultado una clase después de tejerlo al sistema, que aumentará el bytecode del sistema.

— *Dependencia del nombre.* La definición del aspecto depende de las clases del sistema, métodos, atributos, y nombres del argumento, que disminuyen la reutilización de los aspectos. De cualquier modo, existen herramientas que pueden automatizar la definición de los aspectos, aumentando la productividad de los aspectos y la reutilización.

— *Permite utilizar un mismo sistema en diferentes middleware.* La idea de AOP es generar versiones de un sistema incluyendo aspectos. La característica de usar un mismo sistema con diferentes middleware es que se puede conseguir varias versiones del sistema. Sin embargo, esto implica tener varias instancias del sistema (ServerSide) ejecutándose, en vez de uno solo, que afectaría o anularía el control de la concurrencia. En cambio, existe otro patrón DAP que puede solucionar esto fácilmente.



De todas maneras las ventajas superan con creces los pequeños inconvenientes que pueden surgir, por lo que creemos que el uso de PaDA para implementar sistemas distribuidos es realmente ventajoso frente a las técnicas de programación existentes actualmente. Con PaDA podemos beneficiarnos de todas las ventajas que proporciona la programación orientada a aspectos y aplicarla fácilmente a la distribución.

IMPLEMENTACIÓN

La implementación del patrón PaDA está compuesto de cuatro grandes pasos:

— *Identificar los módulos.* Servidor y cliente, mantienen la comunicación distribuida entre ellos.

— *Escribir el aspecto Server-Side.* El aspecto Server-Side es responsable de cambiar el código del módulo server usando la API especifica de distribución, implementándolo accesible a respuestas de llamadas remotas. Este aspecto tiene que cambiar también los módulos que usan como parámetros o valores de retorno del tipo server, depende de la API de distribución. Server-Side maneja las invocaciones a métodos remotos del lado del servidor, por ejemplo, creando el objeto remoto inicial y registrándolo en el servidor de nombres.

— *Escribir el aspecto del Client-Side.* El aspecto Client-Side es responsable de interceptar las llamadas locales originales efectuadas por el cliente y redireccionarlas como llamadas remotas efectuadas al módulo remoto (servidor). En definitiva maneja las invocaciones a métodos remotos del lado del cliente, por ejemplo, adquiriendo la referencia inicial de objeto remoto.

— *Escribir el aspecto manejador de excepciones.* El aspecto manejador de excepciones es responsable de manejar con una nueva excepción añadida por la definición de los aspectos. Estas excepciones creadas envuelven a una excepción no chequeada, lanzándolas sin cambiar la firma del código fuente del sistema original. Por eso, el aspecto manejador de excepciones debe proporcionar también el manejo necesario en las clases del Client-Side.

CONCLUSIÓN Y LÍNEAS FUTURAS

Actualmente, la implementación de distribución en sistemas tiene una alta demanda. La continua evolución a la que se ven sometidos los distintos sistemas desarrollados hace necesaria la implantación de un modelo de desarrollo que permita una fácil adaptabilidad y reutilización de los mismos, sin mencionar la importancia de un mínimo periodo de tiempo para ello.

En este documento se ha enfocado el desarrollo de aplicaciones distribuidas desde la perspectiva del paradigma de la Programación Orientada a Aspectos, concretamente se ha tenido en cuenta el enfoque propuesto por el patrón PaDA donde se implementa en aspectos las especificaciones para los componentes de la aplicación, con el fin de componerlos posteriormente junto al comportamiento funcional para dar lugar a una aplicación distribuida RMI.

Este enfoque ha permitido comprobar que efectivamente la programación orientada a aspectos es una buena aproxima-

ción para el desarrollo de aplicaciones distribuidas ya que una aplicación de estas características dentro de los paradigmas tradicionales como la orientación a objetos aumenta la complejidad de la aplicación. Como los aspectos son compuestos con módulos de aplicación en una aplicación orientada a aspectos, permitiendo la introducción de propiedades adicionales sin comprometer la claridad del código, mantenimiento, reutilización, adaptabilidad, modularidad y escalabilidad del sistema.

La adaptabilidad es una área donde la programación orientada a aspectos sobresale, sin embargo según en que casos de uso será mayor o menor. En el caso de un cambio pequeño y no revolucionario el trabajo requerido será casi por igual en las dos versiones. Si la aplicación original estaba preparada desde su creación inicial para una serie de cambios, está claro que estas consideraciones anticipadas pueden suponer una ventaja a considerar. Ante un cambio rotundo o de gran envergadura, POA es superior con diferencia, ya que con la programación tradicional el sistema entero tendría que ser rehecho.

Pero si tal es la facilidad de agregar, la misma es para eliminar, en el caso de que una aplicación tratada con programación orientada a aspectos decida cambiar su comportamiento eliminando ciertos requisitos, no hace falta rehacer de nuevo la aplicación, simplemente eliminar los aspectos que implementaban dicho requisito. Las aplicaciones se vuelven más flexibles.

Al estar el código de distribución centrado en unos módulos diferenciados, el código está más claro, menos enredado, y en el número de líneas de código de aplicación disminuyen, ya que el código repetitivo es eliminado y concentrado, reduciendo el coste de mantenimiento del código, por la disminución del código está claro que disminuye el mantenimiento pero sobre todo por la claridad y modularidad del código.

La programación orientada a aspectos es el futuro de la programación, es sin duda la quinta generación, no solo soluciona las desventajas de la programación orientada a objetos, sino que deja abierta una puerta con infinidad de soluciones aún por descubrir. Con AspectJ como uno de los lenguajes más fuertes en el momento, se acumulan las ventajas de la orientación a objetos en Java y la orientación a aspectos.

BIBLIOGRAFÍA

(1) Libro: H.M Deitel, P.J. Deitel. Java how to program. Prentice Hall. 2002.

(2) Libro: William Grosso. Java RMI: designing & building distributed applications. O'Reilly & Associates. 2001.

(3) Libro: C.T. Arrington. Enterprise Java with UML. JohnWiley & Sons. 2001.

(4) Libro: Ivan Kiselev. Aspect-Oriented Programming with AspectJ.Sams.2002

(5) Reina A.M. *Visión General de la Programación Orientada a Aspectos*. Informe Técnico del Departamento de Lenguajes y Sistemas Informáticos de la Facultad de Informática y Estadística de la Universidad de Sevilla, Diciembre 2000. http://www.lsi.us.es/~informes/aopv3.pdf

(6) G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier and J. Irwin, *Aspect-Oriented Programming*, Xerox Palo Alto Research Center, 1997.

(7) G. Kiczales, C. Lopes. *Aspect-Oriented Programming with AspectJ TM –Tutorial–*.Xerox Parc. http://www.aspectj.org.

(8) AspectJ TM : *User's Guide*. Xerox Parc Corporation. http://www.aspectj.org

(9) Coady, Y., G. Kiczales, and M. Feeley: Exploring an Aspect-Oriented Approach to Operating System Code. In: Position paper for the Advanced Separation of Concerns Workshop at the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). ACM, Minneapolis, Minnesota, USA (2000).

(10) Clemente P., Hernández J., Sánchez F. *Reutilización y adaptabilidad en componentes distribuidos: ¿es la programación orientada a aspectos una posible solución?* IScDIS'2000. 1er Taller de Trabajo en Ingeniería del Software basada en Componentes Distribuidos, Noviembre 2000. http://tdg.lsi.us.es/~sit02/res/papers/clemente.html

(11) Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J, Irwin, J., *"Aspect-Oriented Programming"* In Proc. ECOOP'97 (Finland, June 1997) Springer-Verlag http://trese.cs.utwente.nl/aop-ecoop99/aop97.html

(12) Sergio Soares and Paulo Borba. PaDA: A Pattern for Distribution Aspects. In Second Latin American Conference on Pattern Languages Programming | SugarLoafPLoP, Itaipava, Rio de Janeiro, Brazil, August 2002. http://www.cin.ufpe.br/~scbs/artigos/soares_borba_sugarloafplop2002.pdf

(13) Sergio Soares, Eduardo Laureano, and Paulo Borba. Implementing distribution and persistence aspects with Aspect J. In Proceedings of OOPSLA'02, Object Oriented Programming Systems Languages and Applications. ACM Press, November 2002. To appear. http://oopsla.acm.org/

(14) Página principal de AspectJ =http://www.aspectj.org visitada 20/11/2002

(15) Página principal de AspectJ= http://www.eclipse.com visitada 26/06/2003

(16) Página del proyecto AspectJ= http://www.parc.xerox.com/spl/projects/aop/aspectj/