Información Tecnológica
Vol. 23(6), 51-62 (2012)
doi: 10.4067/S0718-07642012000600007

Extensión del Diagrama de Secuencias UML (Lenguaje de Modelado Unificado) para el Modelado Orientado a Aspectos

Cristian L. Vidal[(1)], Rodolfo F Schmal[(1)], Sabino Rivero[(1)] y Rodolfo H. Villarroel[(2)]
(1) Escuela de Ingeniería Informática Empresarial, Facultad de Ciencias Empresariales, Universidad de Talca Campus Lircay, Avenida Lircay S/N, Talca-Chile
(e-mail: {cvidal, rschmal, srivero@utalca.cl})
(2) Escuela de Ingeniería Informática, Facultad de Ingeniería, Pontificia Universidad Católica de Valparaíso, Avenida Brasil 2241, Valparaíso-Chile (e-mail: rodolfo.villarroel@ucv.cl)

Recibido Abr. 16, 2012; Aceptado May. 28, 2012; Versión final recibida Jun. 27, 2012

Resumen

Este trabajo revisa los principales elementos de diagrama de secuencia UML (Unified Modeling Language) para modelar el comportamiento dinámico de *software* y describe los pasos para alcanzar un modelado orientado a aspectos con esta herramienta. Esto con el objetivo de lograr una mayor autonomía y "limpieza" en el modelado e implementación de las tareas principales de una aplicación *software*. Para estos efectos, se describe la metodología de desarrollo de *software* orientado a objetos (DSOO) resaltándose la importancia del UML para el modelado de estructuras y comportamiento de aplicaciones *software* orientadas a objetos. Además, se mencionan las principales características de la metodología de desarrollo de *software* orientado a aspectos (DSOA) considerando DSOA como una extensión a DSOO. Esto hace necesario revisar y adaptar el UML para el soporte de modelado orientado a aspectos. En este contexto, se modela un caso de estudio clásico del método de DSOA.

Palabras clave: diagrama de secuencias, UML, DSOA, software, modelado orientado a aspectos

Extension of UML Sequence Diagrams (Unified Modeling Language) to Aspect-Oriented Modeling

Abstract

This article reviews the basic principles of UML sequence diagram (Unified Modeling Language) to model dynamic behavior of software and describes the steps needed to achieve an aspect-oriented modeling using this tool. This with the object of reaching greater autonomy and cleaning during the modeling and implementation of the main tasks of a software application. For these purposes, the object-oriented software development methodology (OOSD) is described remarking the importance of UML for the modeling of structures and behavior of object-oriented software applications. Furthermore, the main characteristics of the aspect-oriented software development methodology (AOSD) are mentioned, considering AOSD as an extension of OOSD. This makes it necessary for reviewing and adapting UML for the support of aspect-oriented modeling. In this context, a classic case study of the AOSD method is modeled.

Keywords: sequence diagrams, UML, AOSD, aspect oriented modeling

INTRODUCCIÓN

El desarrollo de *software* orientado a objetos (DSOO) modulariza los principales elementos estructurales y de comportamiento y las relaciones entre ellos de una aplicación *software*, donde el lenguaje de modelación por defecto es el Lenguaje de Modelamiento Unificado (UML) (Pressman, 2001; Booch et al., 2005). Adicionalmente, como estipula Krechetov et al. (2006), es necesaria la integración de UML con nuevos entornos de desarrollo de software para promover simplicidad y evitar extensiones no necesarias.

UML permite una modelación de los componentes estáticos de una aplicación *software* (diagramas de casos de uso, diagramas de clases), así como del comportamiento dinámico de sus principales elementos durante su funcionamiento (entre ellos diagramas de estados y diagramas de secuencias). Los diagramas de estados permiten la modelación de los principales estados y los eventos que ocasionan sus cambios para una instancia de una clase, o para un sistema como un todo, mientras que los diagramas de secuencias permiten modelar instancias de interacción entre actores u objetos de clases de un sistema a través de mensajes. Mediante estos últimos diagramas es posible conocer lo que ocurre internamente entre los actores e instancias de clases que participan en un diagrama de estados de un sistema de *software*. Según Booch et al. (2005), los diagramas de secuencias UML constituyen elementos relevantes en el desarrollo de una aplicación software.

Para conocer cada uno de los escenarios de funcionamiento de una aplicación y derivar el estado de los objetos y del sistema mismo, los diagramas de secuencias son los indicados. Esta es una justificación pseudo-formal para el uso de diagramas de secuencias UML.

Aún cuando el paradigma de DSOO presenta mejoras considerables respecto a paradigmas anteriores, aún no permite alcanzar una completa modularización ya que persisten elementos que se mezclan (tangling) y se esparcen (scattering) en el desarrollo de una aplicación *software* (Vidal et al., 2012). Para alcanzar una mayor modularización, el paradigma desarrollo de *software* orientado a aspectos (DSOA) representa un nuevo enfoque que separa elementos cruzados (separación de incumbencias) (Vidal et al., 2012), elementos no posibles de independizar en el paradigma de orientación a objetos (Laddad, 2003). En este trabajo, los autores consideran al DSOA como una extensión del DSOO dado que la naturaleza principal de DSOO se mantiene presente en DSOA. Además, según Laddad (2003), DSOA tiene sus orígenes como programación orientada a aspectos, que es una extensión del lenguaje de programación orientado a objetos para alcanzar una mayor modularización.

Por lo tanto es importante adaptar los elementos de DSOO para soportar DSOA en el ciclo de desarrollo de *software*. De esta forma, es necesario acomodar la herramienta de modelación UML a DSOA. En Vidal et al. (2012) se incluye una aplicación de diagramas de casos de uso UML que soporta DSOA, y en el presente trabajo se aplican diagramas de secuencias UML para la modelación del comportamiento dinámico de una aplicación *software* con soporte de DSOA.

En la sección que sigue se describe el lenguaje de modelación UML y en él se presentan los principales elementos y características de diagrama de secuencias UML. A continuación, se presenta una sección con los principales elementos de DSOA, incluyendo la extensión y utilización del diagrama de secuencias UML para modelar DSOA. Posteriormente, se presenta una aplicación de diagramas de secuencias UML con el soporte de DSOA. Finalmente, una sección de conclusiones y referencias.

LENGUAJE DE MODELADO UML

UML es un lenguaje de modelado orientado a objetos que permite representar gráficamente los elementos estáticos y dinámicos de una aplicación *software* (Booch et al., 2005; Pressman, 2001).

En UML un diagrama de secuencias modela la interacción entre los objetos de un sistema de *software* (Booch et al., 2005). Primero, un diagrama de secuencia es etiquetado en un marco con

el operador *sd* y el nombre del diagrama. En él, un objeto es asociado a una caja que incluye una identificación del objeto, la que puede omitirse (objeto anónimo) y la clase del objeto, la que debe ser indicada. Bajo esta caja hay una línea vertical o línea de vida del objeto, y entre líneas de vida tradicionalmente hay mensajes para modelar la comunicación entre objetos. Usualmente hay diferentes tipos de relaciones o instancias de comunicación entre objetos a ser modeladas en un diagrama de secuencias UML mediante el uso de operadores y marcos para generar fragmentos combinados. Adicionalmente, un diagrama de secuencias permite establecer condiciones mediante el uso de banderas. Como ejemplos de operadores y marcos, un diagrama de secuencias UML permite la modelación de comportamiento alternativo u opcional mediante operadores *alt* y *opt* junto a su respectivo marco o fragmento combinado. De la misma manera, para indicar un comportamiento negativo o no esperado se utiliza un fragmento combinado asociado al operador *neg*. Es importante destacar que DSOO incluye el concepto de excepciones para representar un comportamiento no esperado o excepcional en un sistema, modelado mediante un marco y el operador *break*. Además, es posible modelar la entrada de mensajes externos o la salida de mensajes fuera de un marco mediante el uso de puertas (*gates*). Nótese que a través de fragmentos combinados con el uso de marcos, el diagrama de secuencias UML permite una modelación incremental. Los principales elementos de un diagrama de secuencias UML se muestran en la figura 1.

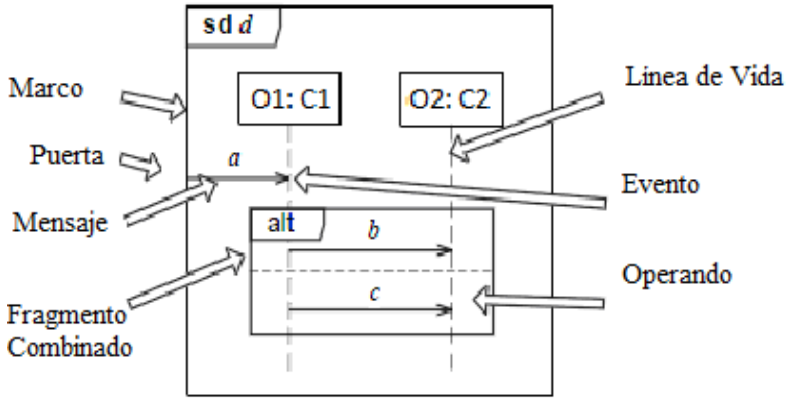

Fig. 1: Principales Elementos de un Diagrama de Secuencias UML.
Fuente: Fleming et al. (2008)

Es relevante indicar que en un diagrama de secuencias existen diferentes tipos de mensajes (Booch et al., 2005): síncronos, asíncronos, plano (sin distinción de sincronía) y mensajes de retorno. En mensajes síncronos, quien lo envía espera por un mensaje de confirmación (para saber si la operación fue exitosa), mientas que en mensajes asíncronos, quienes los envía no requiere esperar la recepción y confirmación de entrega exitosa. Así, quien envía un mensaje asíncrono continúa con su trabajo después de enviado. Un mensaje plano depende de un contexto e interpretación respecto a si es síncrono o no. Un mensaje de retorno representa un mensaje de confirmación cuyo uso es opcional tras la recepción de mensajes asíncronos. La figura 2 ilustra un ejemplo de un diagrama de secuencia con dos objetos anónimos, de clases A y B respectivamente, donde el objeto de clase A envía un mensaje síncrono al objeto de la clase B, y se muestra el uso de flujo continuo en el objeto que envía el mensaje síncrono para esperar por un mensaje de confirmación de su correcta recepción. Nótese que en UML 2.0, el envío de mensaje de confirmación es opcional incluso para mensajes síncronos lo que ilustra la informalidad de este lenguaje de modelación.

Aún cuando hay propuestas de extensión de diagrama de secuencias UML para la modelación de elementos concurrentes (Xie et al, 2007), cabe señalar que UML 2.0 ya permite la modelación de elementos concurrentes junto con ideas de exclusión para diagramas de secuencias (Booch et al, 2005).

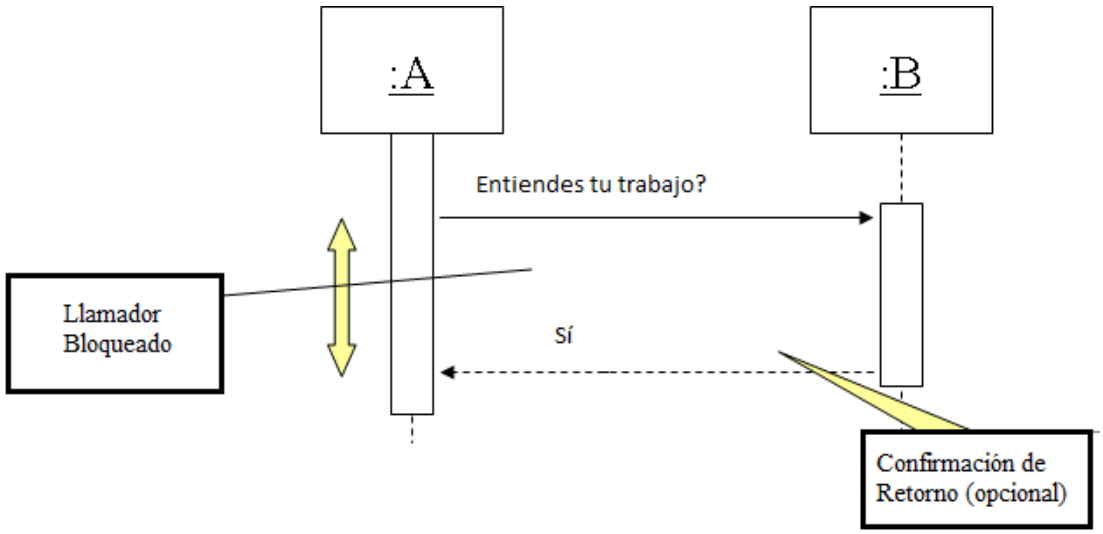

Fig. 2: Mensaje Síncrono y Flujo Continuo en Diagrama de Secuencias UML.

DESARROLLO DE SOFTWARE ORIENTADO A ASPECTOS

DSOA, se basa principalmente en una separación de intereses o incumbencias durante el ciclo de desarrollo de *software* con el objetivo de lograr una mejor abstracción modular del sistema (Tabares et al., 2008; Londoño et al., 2008; Wimmer et al., 2011).

Si bien el paradigma de DSOO presenta una mejor modularización respecto al paradigma de desarrollo de *software* estructurado (Pressman, 2001), persiste la presencia de esparcimiento de funciones (*scattering*) y código enmarañado (*tangling*), lo que explica que en la actualidad el paradigma de orientación a aspectos sea un recurrente tema de investigación para el desarrollo de *software* (Chavez et al., 2006; Vidal et al., 2011; Vidal et al., 2012), dado que éste procura abordar estas complicaciones aún pendientes del paradigma DSOO.

Por *scattering* se hace referencia al esparcimiento de elementos funcionales o no funcionales en diferentes módulos de aplicación. Esto representa un gran problema cuando alguno de estos elementos dispersos cambia porque hace necesaria la alteración en todos los miembros donde se encuentre (Vidal et al., 2012). La figura 3 muestra una funcionalidad que se expande y aparece junto a otras funcionalidades en el desarrollo de una aplicación bancaria web.

El tangling es el fenómeno *que* se produce cuando aparecen diversas funcionalidades que se mezclan con la funcionalidad principal de un módulo (Vidal et al., 2012). Existe un elemento principal o base encargado de la funcionalidad de mayor importancia, y adicionalmente existen elementos que se preocupan de funcionalidades anexas para así lograr el total cumplimiento de la tarea central. Claramente existe una mezcla de funcionalidades, y no existe una separación de tareas, funcionalidades o incumbencias. Hay problemas cuando uno de estos elementos transversales no permite una completa individualización o separación de los módulos que asiste y se requiere la aplicación de un cambio sobre este elemento transversal, ya que este cambio es necesario en cada uno de los módulos donde este elemento aparece. Según Jacobson (2004), si fuera posible lograr una completa separación de tareas (una tarea base no se preocupa de tareas anexas), se obtendría un mejor desarrollo de *software*. En la figura 4 se observa un ejemplo de tangling o mezcla de funcionalidades en el desarrollo de una aplicación bancaria web.

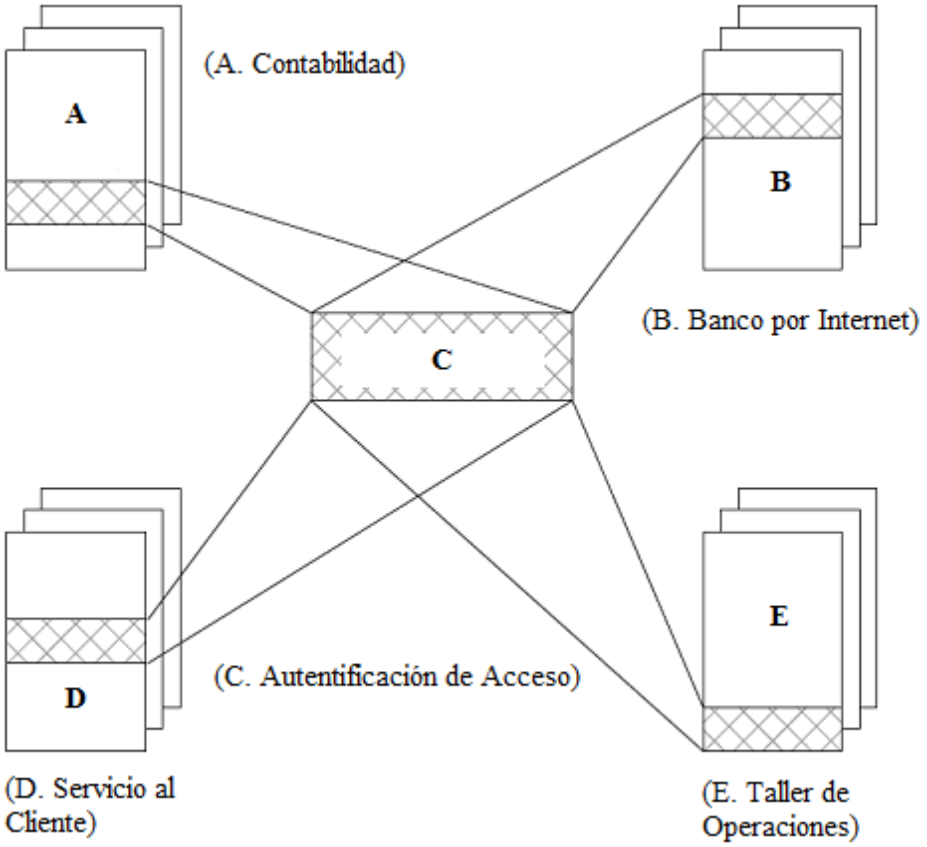

Fig. 3: Ejemplo de *scattering* (Vidal *et al.,* 2011)

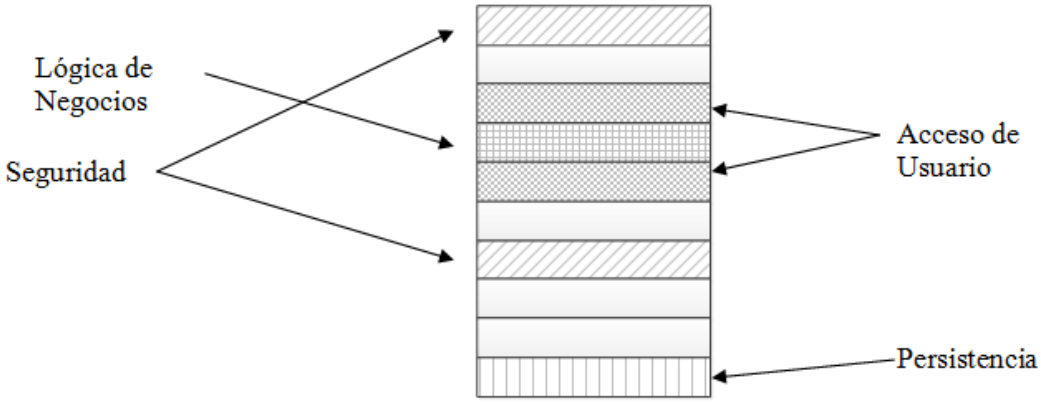

Fig. 4: Ejemplo de *tangling* (Vidal *et al.,* 2011)

Al resolver los problemas de *scattering* y *tangling*, el paradigma DSOA permite una mayor modularización, lo que se traduce en elementos más limpios y fáciles de entender, especificar, diseñar e implementar, obteniéndose un *software* más mantenible, extensible y reutilizable.

Antes de revisar ideas de modelación orientada a aspectos mediante el uso de diagramas de secuencias UML, es necesario indicar la relación existente entre los conceptos de aspectos o funcionalidades cruzadas posibles de aislar en DSOA, y el concepto de excepción. Una excepción es un elemento que se utiliza para indicar situaciones no previsibles, que usualmente son utilizadas en diversas partes de la aplicación para manejar estas situaciones “excepcionales” en una aplicación orientada a objeto. Así, el manejo de excepciones se dispersa y se mezcla con otras funcionalidades en cada uno de los métodos donde ellas se incluyen. Si fuera posible modelar separadamente excepciones en diagramas de secuencias UML, éstas serían una

extensión de ellos para soportar DSOA dado que una excepción tiene la naturaleza base de un aspecto: funcionalidad cruzada imposible de separar con los paradigmas previos a DSOA. Una excepción, se modela con el uso de un marco y un operador "break" en diagrama de secuencias tradicional, sin embargo esta forma de modelar no representa la base de separación de incumbencias de DSOA. A continuación se revisan extensiones existentes de diagrama de secuencias UML para soportar DSOA, y se adicionan elementos para una mejor percepción de este paradigma.

Una noción esencial del paradigma DSOA es la separación de funcionalidades cruzadas para que así las funcionalidades principales estén directamente involucradas en sus tareas, y aquellas funcionalidades cruzadas estén separadas y conozcan el momento y condiciones válidas para su aparición (Vidal et al., 2011; Vidal et al., 2012).

Respecto al uso de UML y aspectos, Suzuki y Yamamoto (1999) proponen una extensión de UML para soportar aspectos. Ellos extienden el metamodelo de UML y un aspecto es incorporado como elemento del metamodelo derivado de *Classifier*. El modelado de los aspectos lo realizan utilizando diagramas de clases y paquetes de UML. Similarmente, Anjum (2006) propone el uso de diagramas de clases para modelar los requerimientos no funcionales como aspectos. Sin embargo, las dos propuestas mencionadas no utilizan diagramas de secuencia para soportar DSOA.

Es importante señalar que ya hay trabajos acerca de UML y diagramas de secuencias para soportar DSOA entre ellos Whittle et al. (2007), Zhang et al. (2009), Gupta et al. (2011), Wimmer et al. (2011), y Tabares et al. (2008), cuyos principales elementos respecto a modelación orientada a aspectos y diagramas de secuencias UML se muestran en la tabla 1.

Tabla 1: Trabajos de UML orientados a aspectos.

Whittle et al. (2007)	muestra una adaptación global de UML para soportar DSOA sin incluir una descripción de la extensión misma para diagramas de secuencias
Zhang et al. (2009)	captura nociones de DSOA, pero no captura un elemento principal de este paradigma que es la noción de individualismo o aislamiento de elementos principales.
Gupta et al. (2011)	presenta extensiones generales de UML, mencionando extensiones para diagramas de comunicación, secuencias y colaboración, pero sin detallar el uso de diagramas de secuencias orientados a aspectos.
Wimmer et al. (2011)	presenta una colección de trabajos ya realizados para el soporte de DSOA en UML entre los cuales se destaca el uso de diagramas de secuencias UML con extensión a orientación a aspectos las que son similares a las descritas en la siguiente sección.
Tabares et al. (2008)	muestra el uso de diagramas de secuencias UML con el uso de estereotipos y comentarios. Sin embargo, similar a Zhang et al. (2009) hay una invocación explícita del comportamiento de aspectos.

Desde una visión de orientación a aspectos, los elementos principales de una aplicación *software* deben trabajar sólo con sus funciones principales y los aspectos encargarse de realizar tareas anexas a estas labores sin una explícita solicitud por ese trabajo. En la figura 5 se muestran dos objetos donde uno de ellos es un aspecto (objeto *back-check* de la clase *beforeAspect*), y es quien recibe el mensaje del objeto principal (*purchase* plan de la clase *class*) para realizar su función. Esto, claramente, no respeta el principio de aislamiento de elementos principales para una completa separación de incumbencias, aún cuando la idea de representar un aspecto como una instancia de un objeto, así como el momento cuando éste debe aparecer están claramente presentes (uso de *before*: antes, *after*: después, *around*: durante).

Además, es necesario destacar que un aspecto se modela en forma similar a un participante de un diagrama de secuencias UML, actor u objeto de una clase del sistema. La naturaleza de un aspecto es análoga a la de un objeto que debe reaccionar ante circunstancias especiales antes

(*before*), después (*after*) o durante (*around)* tales circunstancias. Sin embargo, de cada aspecto habría sólo una instancia encargada de trabajar con los participantes del sistema.
Dado que el aislamiento de los principales elementos de DSOA no es respetado en la figura 5, a partir de ideas de Tabares et al. (2008) y Wimmer et al. (2011), un diagrama de secuencias UML orientado a aspectos debe considerar:

(i). el uso de instancias de aspecto *beforeAspect*, *AfterAspect*, y *AroundAspect*.

(ii). los aspectos encargados de realizar sus labores sin una invocación previa, dado que ellos pertenecen al contexto del objeto del punto de unión (*joinpoint*), y pueden acceder a la información permitida y solicitar información restrictiva del objeto si fuera necesario. Cuando el aspecto termina de realizar su labor, sus resultados pueden ser accesibles por el objeto principal.

(iii). los confines de modelación, no de implementación, deben incluir un mensaje para indicar la ocurrencia de un punto de unión desde la instancia en el punto de corte y el aspecto asociado. Este mensaje incluye el uso de un estereotipo <<*JoinPoint*>> para referenciar la presencia de un punto de corte.

Nótese que en i, ii, y iii se asume una definición previa de aspectos junto con sus puntos de corte (*pointcuts* = conjunto de puntos de unión o *joinpoints*), su comportamiento (*advices before*, *after*, y *around*) y ámbito respecto a la información accesible de los objetos principales que ocasionan la reacción del aspecto.

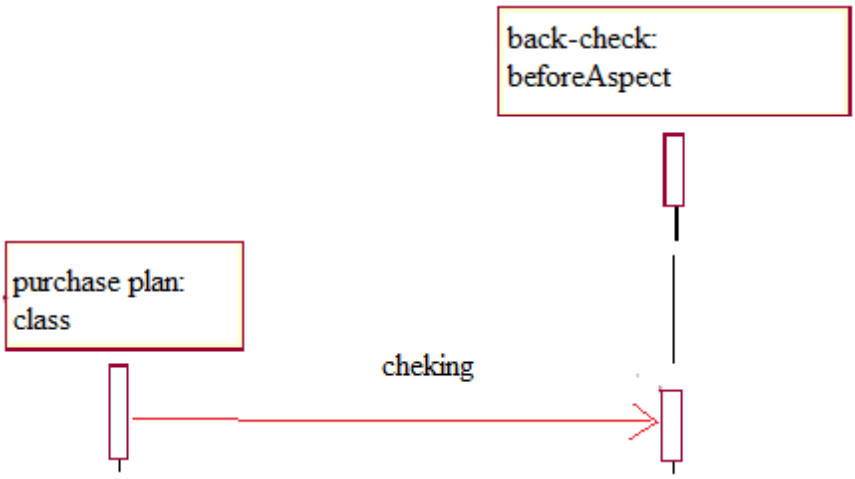

Fig. 5: Ejemplo de Previa Extensión de Diagrama de Secuencias UML para DSOA
Fuente: Elaboración propia a partir de Zhang et al. (2009)

Es claro que el mensaje para avisar la ocurrencia de un punto de unión es no directo en la práctica, ya que en la definición de un aspecto debe explicitarse cuando este debería aparecer junto al trabajo del mismo. Sin embargo, dado cualquier comportamiento determinístico o no, es posible inferir situaciones dada la invocación de un método base de puntos de corte. Aún así, para entender la presencia de un aspecto, es necesaria la inclusión de este estereotipo.

La figura 6 presenta una adaptación de la figura 5 que soporta esta propuesta de extensión de un diagrama de secuencias UML, donde se asume que la labor del aspecto es informar acerca de la disponibilidad de un ítem. Aplicando esta extensión, cuando el objeto principal desea obtener ítems lo que representa un punto de unión, el aspecto reacciona y envía un mensaje acerca de la disponibilidad o no del ítem requerido. Se asume que el aspecto tiene acceso a la identificación del ítem a ser chequeado, dada la definición de ámbito y contexto respecto a las clases relacionadas para así acceder a información total o parcial de las instancias de las clases. De esta manera, dada la presencia de un punto de unión, el aspecto es el encargado de reconocerlo y realizar sus labores, y enviar resultados al objeto principal.

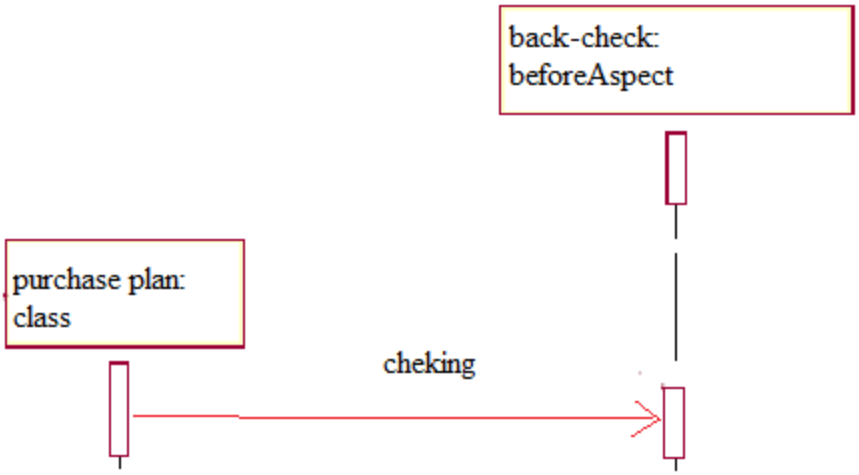

Fig. 6: Aplicación de Propuesta de Diagrama de Secuencia Orientado a Aspectos.

APLICACIÓN DE DIAGRAMA DE SECUENCIAS UML Y ORIENTACIÓN A ASPECTOS

Un ejemplo clásico de orientación a aspectos (Vidal et al., 2012) es el ingreso de usuarios a un sistema de *software*, y la aparición de un servicio de autentificación de usuarios o *login* que representa un aspecto before (antes de ingresar al sistema, el usuario debe identificarse). En este caso, se supone que el aspecto reconoce cuando el usuario desea ingresar al sistema y aparece inmediatamente para solicitar información de autentificación. Como se aprecia, en este ejemplo el aspecto debe solicitar esta información explícitamente. La figura 7 muestra el diagrama de casos de uso orientado a aspectos asociado a esta aplicación.

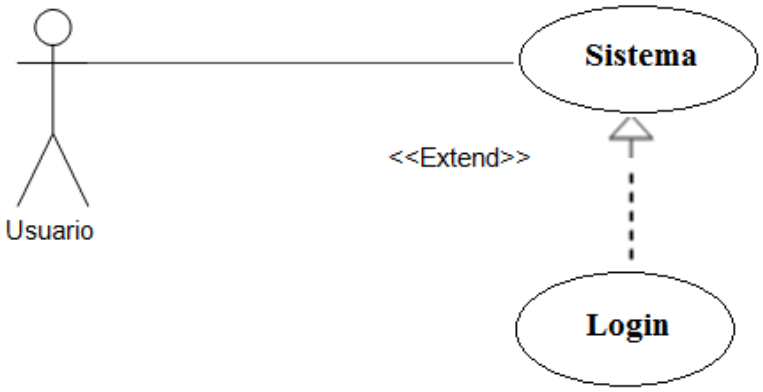

Fig. 7: Ejemplo de Caso de Uso Orientado a Aspectos.
Fuente: Vidal et al. (2012)

Para esta modelación se asume una clase Usuario, una clase Sistema, un aspecto *Login*, y en el diagrama de secuencia asociado, hay instancias de estos 3 elementos.

Las figuras 8 y 9 presentan los dos posibles escenarios: cuando el usuario entrega información correcta y tiene acceso al sistema, y cuando el usuario es no válido. Se asume que el aspecto es quien reconoce la idea del usuario de ingresar al sistema e invoca un método del usuario denominado IngresarInformaciónAutentificación(). A continuación, el usuario se comunica con el aspecto y entrega Id y Clave al método EntregarInformación(Id, Clave) del aspecto *Login*. En este mensaje es donde se producen los dos escenarios que se muestran en las figuras 8 y 9 respectivamente. En la figura 7, cuando el usuario recibe un mensaje de AccesoVálido, el usuario tiene acceso al sistema e invoca los métodos del sistema Operación1(...), Operación2(…) y así sucesivamente. Por otro lado, en la figura 8 se muestra el caso cuando el usuario no es válido, y en caso de querer ingresar nuevamente al sistema, debe repetir el proceso de identificación asociado. En ambas figuras, los diagramas de secuencias orientado a aspectos capturan los

escenarios posibles, observándose la completa autonomía de los elementos principales del sistema, y por consecuencia se logra una mayor separación de incumbencias que usando la propuesta de diagrama de secuencias orientado a aspectos de Zhang et al. (2009).

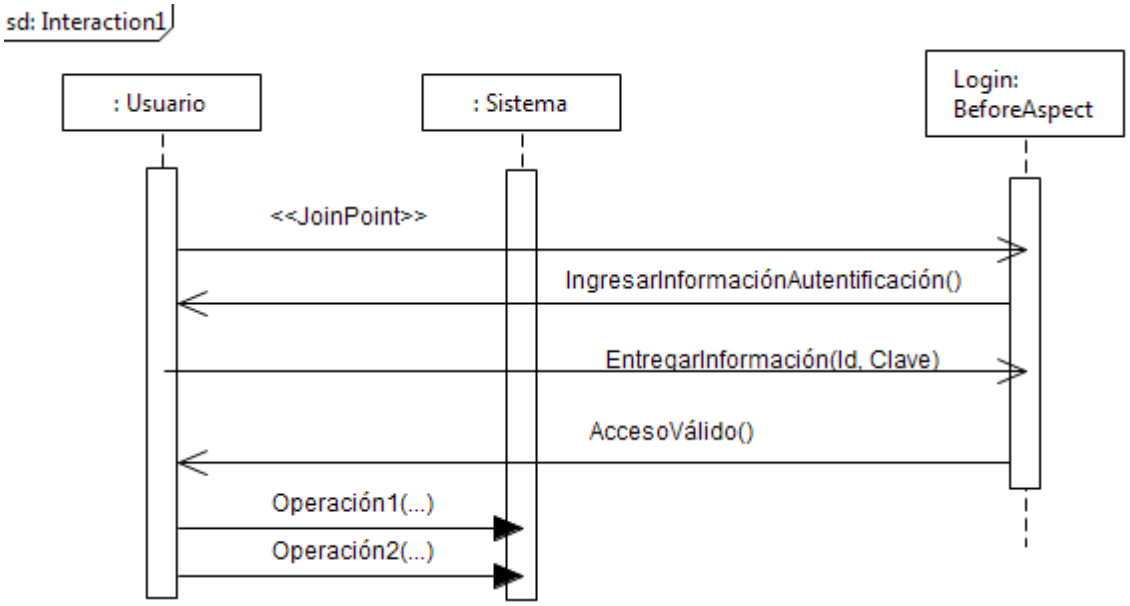

Fig. 8: Aplicación de Extensión de Diagrama de Secuencia UML Orientado a Aspectos para Ingreso a Sistema Caso 1: Autentificación Válida.

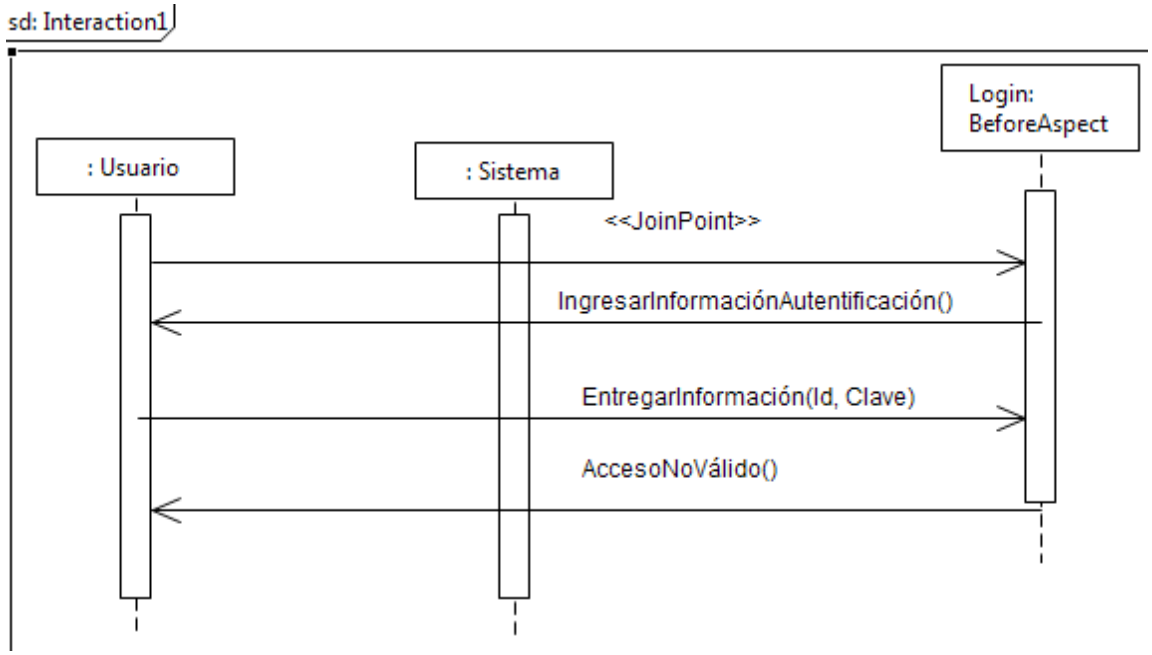

Fig. 9: Aplicación de Extensión de Diagrama de Secuencia UML Orientado a Aspectos para Ingreso a Sistema Caso 2: Autentificación No Válida.

CONCLUSIONES

El diagrama de secuencia UML permite una modelación en detalle de los escenarios en el funcionamiento de una aplicación , incluyendo a los actores de la misma (objetos). Tras presentar las deficiencias de *scattering* y *tangling* aún presentes en el paradigma DSOO, se revisaron las características de DSOA destinadas a disminuir o eliminar estas deficiencias.

Este trabajo revisó, extendió y aplicó diagramas de secuencias UML para soportar DSOA. Claramente, DSOA permite una mayor y/o completa separación de incumbencias logrando mayor autonomía o "limpieza" en las funcionalidades principales de una aplicación *software* ya que éstas sólo están preocupadas de su función principal, dado que los aspectos o funcionalidades transversales son los encargados de las tareas anexas (dónde y cuándo aparecer, y qué resultados entregar). La extensión de diagrama de secuencias orientado a aspectos propuesta en este trabajo, permite lograr una mayor individualización de las tareas principales. De esta forma, los aspectos son los encargados de reaccionar ante situaciones que los requieran.

Como trabajo futuro, se está trabajando en la aplicación de diagramas de secuencias orientado a aspectos para modelar la tolerancia a fallos en sistemas distribuidos. En este contexto, la modelación de elementos concurrentes y distribuidos tiene gran importancia. Adicionalmente, se trabajará en una extensión de herramientas libres para el soporte de UML orientado a aspectos.

REFERENCIAS

Anjum, S. Incorporating non-functional requirements with UML models. Thesis requirement for the degree of Master of Applied Science in Electrical and Computer Engineering. University of Waterloo, Waterloo, Canada (2006).

Booch, G., Rumbaugh, J. y Jacobson, I. , Unified Modeling Language User Guide, Segunda Edición , Addison-Wesley Object Technology Series (2005).

Bowen, J., Formal Specification and Documentation using Z: A Case Study Approach, Center for Applied Formal Methods, London South Bank University (1996).

Chavez, C., Garcia, A., Kulesza, U., Sant Anna, C. y Lucena, C., Crosscutting Interfaces for Aspect-Oriented Modeling, Journal of the Brazilian Computer Society, 12(1), 43-58, Junio (2006).

Fleming, S., Kraemer, E., Stirewalt, R. E. K., Xie, S., and Dillon, L. K., A Study of Student Strategies for the Corrective Maintenance of Concurrent Software, In Proceedings of the IEEE International Conference on Software Engineering, ICSE'08, Leipzig, Germany, Mayo (2008).

Gupta, P., Garg, S. y Khalon, K.S., Designing Aspects Using Various UML Diagrams in Resource-Pool Management, International Journal of Advanced Engineering Sciences and Technologies, IJAEST, 7(2), 228-233 (2011).

Krechetov, I., Tekinerdogan, B. y Garcia, A., Towards an Integrated Aspect-Oriented Modeling Approach for Software Architecture Design, 8th International Workshop on Aspect-Oriented Modeling at AOSD Conference, Bonn, Germany, Marzo (2006).

Jacobson, I., Aspect-Oriented Software Development with Use Cases, Addison Wesley Professional, Upper Saddle River, NJ, USA, (2004).

Laddad, R., AspecJ in Action, Practical Aspect-Oriented Programming, Manning Publications Co., Londres, Inglaterra (2003).

Londoño, L.F., Anaya, R. y Tabares, M. S., Análisis de la Ingeniería de Requisitos Orientada por Aspectos según la Industria del Software, Revista EIA, Escuela de Ingeniería de Antioquia, n. 9, 43-52, Julio (2008).

Pressman, R., Ingeniería del Software - Un enfoque práctico, 5ª Edición, McGraw Hill (2001).

Suzuki, J., y Yamamoto, Y. Extending UML with aspects: Aspect support in the design phase. Lecture Notes in Computer Sciences: Proceedings of the Workshop on object-oriented Technology, 299-300, (1999).

Tabares, M. S., Alferez, G. H. y Alferez, E. M., El desarrollo de software orientado a aspectos: un caso práctico para un sistema de ayuda en línea, Revista avances en sistemas e informática, 5(2), 61-68 Junio (2008).

Vidal, C., Hernández, D., Gutiérrez, C., Meza, R. y López, L., Modelación Formal Orientada a Aspectos Usando AspectZ, Encuentro Chileno de Computación 2011, Curicó, Chile (2011).

Vidal, C., Hernández, D., Pereira, C. y Del Río, M., Aplicación de Modelación Orientada a Aspectos, Información Tecnológica, 23(1), 3-12 (2012).

Villarroel, R., Fernández-Medina, E., Piattini, M. y Trujillo, J., A UML 2.0/OCL Extension for Designing Secure Data Warehouses, Journal of Research and Practice in Information Technology, 38(1), (2006).

Whittle, J. y Jayaraman, P., MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation, Models in Software Engineering, Springer-Verlag, Berlin, 16-27 (2008).

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

Xie, S., Kraemer, E., Stirewalt, R. E. K., Empirical evaluation of a UML sequence diagram with adornments to support understanding of thread interactions, In Proceeding of the 15th IEEE International Conference on Program Comprehension, ICPC'07, (2007).

Zhang, J., Liu, Y. C. G., Li, H., Using Sequence Diagram to support Aspect-Oriented Programming in MDA, In Procedding of International Conference on Intelligent Human-Machine Systems and Cybernetic, IHMSC'09, Hangzhou, China, August 26-27 (2009).