Extensión del Diagrama de Secuencias UML para el Modelado orientado a Aspectos

Cristian L. Vidal⁽¹⁾, Leopoldo P. López⁽²⁾, Sabino E. Rivero⁽³⁾ y Rogelio O. Meza⁽³⁾

(1) Universidad de Talca, Escuela de Ingeniería Informática Empresarial, Facultad de Ciencias Empresariales, Campus Lircay, Avenida Lircay S/N, Talca-Chile (e-mail: cvidal@utalca.cl; srivero@utalca.cl)

(2) Universidad de Talca, Instituto de Investigación y Desarrollo Educacional, IIIDE, Campus Lircay, Avenida Lircay S/N, Talca-Chile (e-mail: llopez@utalca.cl)

(3) Universidad de Talca, Escuela de Ingeniería Civil en Computación, Facultad de Ingeniería, Campus Los Niches, KM 1, Curicó-Chile (e-mail: rmeza@leaf.cl)

Recibido Ene. 29, 2013; Aceptado Mar. 18, 2013; Versión final recibida Jul. 29, 2013

Resumen

EL objetivo principal de este artículo es revisar los diagramas de secuencia en el lenguaje unificado de modelado UML, presentando STAIRS como una alternativa para formalizar los diagramas de interacción en UML, en especial los diagramas de secuencia. STAIRS es un lenguaje formal para modelar diagramas interactivos de UML, especialmente diagramas de secuencia. Se analizan los problemas intrínsecos de ambigüedad que tiene UML y que producen modelos no necesariamente correctos. Se muestra que STAIRS representa una solución que hace aportes para disminuir dicha ambigüedad. Se concluye que aunque existen nuevos enfoques para el desarrollo de software, como por ejemplo desarrollo orientado a aspectos, UML se adapta a estos nuevos desarrollos. El análisis y revisión presentados en este artículo puede ser de especial ayuda para clarificar estos temas.

Palabras clave: STAIRS, UML, diagramas de secuencia, modelado orientado a aspectos, desarrollo de software

Extension of UML Sequence Diagrams for Aspect-Oriented Modeling

Abstract

The objective of this paper is to review the unified modeling language UML sequence diagrams, presenting STAIRS as an alternative way of formalizing UML interaction diagrams, especially sequence diagrams. STAIRS is a formal language for modeling interactive UML diagrams, especially sequence diagrams. The inherent problems of ambiguity that UML models have, including the production of models that are not necessarily corrected, are analyzed. It is demonstrated that STAIRS contributes to reducing this ambiguity. It is concluded that although there are new approaches for software development, such as aspect-oriented development, UML is capable of adapting to these developments. The review and analysis presented in this article could be especially useful for clarifying these aspects.

Keywords: STAIRS, UML, sequence diagrams, aspect-oriented modeling, software development

INTRODUCCIÓN

De acuerdo a la historia del software y la computación (Somerville, 2004), el proceso de desarrollo de software ha evolucionado desde un modelo secuencial o cascada hasta los actuales modelos iterativos e incrementales. Esta evolución también está presente en las diversas metodologías de desarrollo de software: desde los estructurados e imperativos hasta el orientado a objetos. Independiente de la metodología, hay cinco fases no necesariamente secuenciales en un proceso de desarrollo de software: definición de requisitos, diseño de software, implementación, integración y testing, y operación y mantenimiento (Somerville, 2004).

Sin importar la metodología de desarrollo de software escogida, el modelamiento formal permite obtener una comprensión más profunda del proceso de desarrollo ya que las matemáticas exigen consistencia y exactitud. Las matemáticas usualmente son aplicadas en las primeras etapas del desarrollo de software. Actualmente, la metodología de desarrollo de software más utilizada es la orientación a objetos, la que cuenta con una diversidad de lenguajes, herramientas de programación y de diseño. Respecto a los lenguajes de diseño orientado a objetos, UML es casi por defecto el lenguaje de modelado. UML ofrece diferentes perspectivas para un modelo de software como por ejemplo diagramas de casos de uso, diagramas de clases y diagramas de secuencia. Con respecto a UML y el modelamiento formal, STAIRS (Haugen et al., 2003; Kobro et al., 2009; Runde et al., 2007) permite especificar formalmente diagramas de secuencia UML (Fakhroutdinov, 2012; Kobro et al., 2009).

EL objetivo principal de este artículo es revisar los diagramas de secuencia en UML, presentando STAIRS como una alternativa para formalizar los diagramas de interacción en UML, en especial los diagramas de secuencia buscando trabajos futuros que estén relacionados. Al final este artículo presenta ideas para extender los diagramas de secuencia y capturar los fundamentos del desarrollo de software basado en el DSOA, mencionando el trabajo futuro para extender STAIRS para soportar dicha metodología (Aspect-STAIRS), además de desarrollar una herramienta para dichas extensiones.

DIAGRAMAS DE SECUENCIA UML

Un diagrama de secuencia permite modelar la interacción entre los objetos de un software que será modelado (Booch et al., 2005; Fakhroutdinov, 2012; Kobro et al., 2009). Un diagrama de secuencia tradicionalmente modela sólo una parte del software. En un diagrama de secuencia, un objeto está asociado a una caja que incluye su identificación y su clase. En esta caja, la clase del objeto debe ser indicada, pero su nombre puede ser omitido. Un objeto sin nombre es un *objeto anónimo*. Además, bajo el objeto hay una línea de tiempo, una línea vertical, expresando su tiempo de vida. Para modelar la comunicación entre objetos, un diagrama de secuencia usa mensajes. Además, hay diferentes tipos de relaciones para modelar objetos y sus interacciones. En este sentido, un diagrama de secuencia permite un modelamiento incremental con el uso de *frames*. Por otro lado, para establecer una condición sobre un comportamiento se usan *guards*. Los comportamientos alternativos u opcionales son modelados por operadores como *alt* y *opt* y los comportamientos negativos son modelados por el operador *neg*. Las siguientes secciones describen los mensajes y fragmentos combinados (frames) que son la base del modelamiento composicional de diagramas de secuencia UML. En general, los principales elementos de un diagrama de secuencia se muestran en la Figura 1 (Fleming et al., 2008). Los principales elementos de un diagrama de secuencias UML se detallan en lo que sigue.

Fig. 1: Elementos Principales de un Diagrama de Secuencia UML.

Mensajes: Los objetos en un diagrama de secuencia se comunican usando mensajes. Hay una clara diferencia entre enviar y recibir mensajes. La Figura 2 (Fakhroutdinov, 2012; Kobro et al., 2009) muestra que hay diferentes tipos de mensajes con una clara diferencia entre síncrono, asíncrono, mensajes planos y retornos. Un mensaje síncrono implica que el transmisor espera hasta que el mensaje es recibido y recibe una confirmación satisfactoria de la recepción desde el receptor. Por lo tanto, el transmisor se detiene esperando por una operación completa y satisfactoria. Por otro lado, un mensaje asíncrono no necesita esperar por una confirmación o recepción satisfactoria. Entonces, el transmisor luego de enviar un mensaje asíncrono continúa haciendo su trabajo. Un mensaje plano representa un mensaje síncrono o asíncrono dependiendo de la interpretación o el contexto. UML, que sin OCL (Soeken et al., 2010) no permite formalizar diagramas, es posible usar mensajes asíncronos, síncronos y planos. Además, en un diagrama de secuencia, un mensaje de retorno es usado para mostrar un mensaje de confirmación. La Figura 3 muestra un ejemplo de un diagrama de secuencia con dos objetos anónimos de clases A y B respectivamente, en donde el objeto de clase A envía un mensaje asíncrono al objeto de clase B. Esta figura también muestra el uso de un flujo anidado con que el transmisor espera por el mensaje de confirmación de una correcta recepción del mensaje. De acuerdo a UML 2.0 (Fakhroutdinov, 2012; Kobro et al., 2009), el mensaje de confirmación es opcional. Esta flexibilidad pone de manifiesto la no formalidad de UML.

Marcos y Fragmentos Combinados: Los marcos en un diagrama de secuencia son análogos a una función para el desarrollo de software estructurado y a un objeto para el desarrollo de software orientado a objetos. Los marcos permiten encapsular diagramas de secuencia y reusarlos como parte de un diagrama de secuencia más grande. Por lo tanto es posible modelar y detallar el comportamiento básico en un diagrama de secuencia A y después incorporarlo como parte de otro diagrama de secuencia B sin remodelar A o preocuparse de los detalles. Con el uso de marcos, los diagramas de secuencia UML permiten modularización e integración del comportamiento. Así, los marcos permiten modelar el comportamiento de un sistema de manera separada y también integrar comportamientos ya modelados. La Figura 4 (Kobro et al., 2009) muestra la estructura genérica de un frame en un diagrama de secuencia y la Figura 5 (Kobro et al., 2009) muestra un ejemplo general de integración de un frame como parte de otro diagrama de secuencia. Esta situación se denomina fragmento combinado.

Fig. 2: Tipos de mensajes soportados por un diagrama de secuencia

Fig. 3: Mensaje síncrono y flujo anidado en un diagrama de secuencia.

Fig. 4: Estructura de un marco en un diagrama de secuencia UML.

Fig. 5: Uso de un marco existente como parte de otro diagrama de secuencia.

En general, los marcos en UML permiten un modelado composicional y son usados en los diagramas de secuencia con este propósito. La forma de diferenciar marcos entre distintos modelos UML o vistas, además de su contenido es incluir una palabra clave como la primera palabra de la etiqueta como por ejemplo *sd* para diagramas de secuencia y *act* para diagramas de actividad. Las compuertas son otro elemento importante de un diagrama de secuencia, ya que permiten enviar mensajes hacia el exterior y recibir mensajes externos. Sin embargo, los marcos también permiten enviar mensajes hasta su frontera y obtener mensajes desde ella.

En el contexto de STAIRS, es importante conocer las bases de los operadores alt y neg. Alt permite escoger comportamientos alternativos pero no obligatorios. Por lo tanto, los desarrolladores software pueden entender que cualquiera de las opciones de alt es suficiente para un funcionamiento correcto del sistema. Por medio del operador Alt se puede indicar un comportamiento potencial pero también un comportamiento obligatorio. La única forma de notar la diferencia es por medio de un mensaje (etiqueta). De la misma manera que los problemas con alt, existen problemas con el operador neg. Con neg, es posible indicar el comportamiento como negativo al principio y después cambia a positivo. La idea de comportamiento positivo y negativo está presentadas en la siguiente sección.

UML 2.0 permite modelar recursividad, secciones críticas y compartir (Fakhroutdinov, 2012). Por otra parte (Fleming et al., 2008; Xie et al., 2007), presentan una extensión a los diagramas de secuencia para soportar recursividad y secciones críticas aunque son ideas incluidas previamente en la versión de UML. Sin embargo, el objetivo principal (Fleming et al., 2008; Xie et al., 2007) es alcanzar una mejor comprensión del concepto de concurrencia y sus detalles asociados en un entorno académico. Esta propuesta usa el color verde para identificar un objeto o hilo que ya está usando un recurso exclusivo, amarillo para identificar un objeto que busca usar un recurso exclusivo que está siendo usado y rojo cuando un objeto amarillo intenta acceder a un recurso exclusivo que está siendo usado y está bloqueado. La figura 6 (Fakhroutdinov, 2012) muestra un ejemplo de un diagrama de secuencia usando fragmentos combinados para paralelismo (par) y secciones críticas (critical). La figura 7 muestra un ejemplo de la extensión propuesta a los diagramas de secuencia (Xie et al., 2007).

Fig. 6: Ejemplo de uso de fragmentos combinados.

Fig. 7: Ejemplo de la propuesta de extensión de los diagramas de secuencia para soportar concurrencia.

REFINAMIENTO

El refinamiento está presente en los actuales procesos de desarrollo de software como por ejemplo desarrollo ágil, evolutivo, RUP (Proceso Unificado de Rational) y XP (Programación Extrema) (Somerville, 2004). La idea principal del refinamiento es, dada una solución como un estado base sin incluir todos los detalles del sistema, mejorar esta solución agregando acciones y eventos y reclasificando algunos de ellos también. En el proceso del refinamiento, todas las acciones o eventos relacionados al sistema están categorizadas como positivas, y como negativas las acciones soportadas y aceptadas por el sistema, las acciones que nunca sucederán o serán soportadas por el sistema. Las acciones inconclusas, son las que no son consideradas o analizadas todavía (Haugen et al., 2003). De acuerdo a (Haugen et al., 2003; Fleming et al., 2008; Haugen et al., 2005), el proceso de refinamiento se separa en tres categorías: complemento, reducción y detalle. Complemento significa agregar más características no consideradas todavía como

positivas o negativas. La reducción significa corregir o restablecer características previamente consideradas positivas y reasignarlas como características negativas. El detalle significa obtener más detalles del modelado como por ejemplo agregar guardias, redefinir algunos tipos de mensajes, cambiar la firma de los mensajes entre otros. La Figura 8 (Xie et al., 2007) presenta un ejemplo de un diagrama de secuencia usando fragmentos combinados alt y neg para identificar alternativas a un comportamiento negativo en UML. Indudablemente y de acuerdo a (Haugen et al., 2003; Fleming et al., 2008; Haugen et al., 2005), el refinamiento es un principio básico de STAIRS tal como será descrito en la siguiente sección.

Fig. 8: Uso de fragmentos combinados.

STAIRS

STAIRS es un lenguaje formal para un desarrollo composicional en UML específicamente con diagramas de secuencia UML (Haugen et al., 2003; Ragnhild, 2007), específicamente para formalizar el comportamiento dinámico y las interacciones. STAIRS fue propuesto en el trabajo "STAIRS - Steps to Analyze Interactions with Refinement Semantics" (Haugen et al., 2003) y hasta ahora hay extensiones para modelar concurrencia (Haugen et al., 2003) y guardias (Kobro et al., 2005) y también para aplicar este lenguaje formal (Kobro et al., 2009; Haugen et al., 2005; Soldal, 2007). STAIRS significa Steps to Analyze Interactions with Refinement Semantics (Haugen et al., 2003).

Conceptos Generales de STAIRS

UML, como un lenguaje informal, en cada tipo de modelo ofrece diferentes items para modelar, los cuales están sujetos a la interpretación e ideas del modelador. Por lo tanto, sin una clara explicación textual, obtener la misma interpretación e idea no es una tarea fácil bajo especificaciones (Haugen et al., 2003; Soldal, 2007; Runde et al., 2007). Especialmente en los diagramas de secuencia, hay items para especificar fragmentos combinados los cuales permiten interpretaciones ambiguas. Originalmente STAIRS (Fleming et al., 2008) propone soluciones para los elementos alt y neg de los diagramas de secuencia. Estos ítems a menudo indican un comportamiento potencial, pero no obligatorio y STAIRS ofrece items para definir fragmentos combinados para indicar explícitamente un comportamiento obligatorio. Claramente STAIRS es un lenguaje formal y ayuda a solucionar problemas de modelación. Por otro lado, STAIRS acota la riqueza de UML.

Los mensajes son el elemento más común de los diagramas de secuencia, y representan comportamiento. STAIRS distingue entre comportamiento positivo, negativo e inconcluso (Haugen et al., 2003; Soldal, 2007; Runde et al., 2007). La idea principal del refinamiento usando STAIRS es definir primero un comportamiento básico, ya sea positivo, negativo o inconcluso, luego reducir los comportamientos inconclusos cambiándolos a positivo o negativo, agregando más comportamiento negativo redefiniendo el comportamiento positivo y agregando nuevos detalles al diagrama de secuencia lo que no significa directamente un nuevo

comportamiento como cambiar de asíncrono a síncrono o vice-versa, agregando nuevos parámetros o guardias al mensaje y así sucesivamente. Formalmente, la sección Refinamiento Formal y sus sub-secciones detallan estos tres principios del refinamiento usando STAIRS.

La Figura 9 muestra H , un conjunto de comportamientos de un sistema con las tres divisiones señaladas: positivo p , negativo n e inconcluso, la diferencia entre H y la unión de p y n , $H \setminus (p \cup n)$. STAIRS busca obtener una especificación completa de un sistema mediante el refinamiento disminuyendo el comportamiento inconcluso, estableciendo correctamente el comportamiento positivo y definiendo claramente el comportamiento negativo tal que no pueda cambiar su condición.

Fig. 9: Clasificación del comportamiento de un software.

Debido a que STAIRS es un lenguaje formal, la semántica de STAIRS está detallada en (Haugen et al., 2003; Runde et al., 2007) y se muestra en la siguiente sub-sección. Este artículo, en la sección Refinamiento Formal describe la semántica de los operadores *xalt* y *refuse*, los que permiten modelar el comportamiento obligatorio con tal de evitar interpretaciones ambiguas.

Sintaxis BNF (Forma Backus-Naur)

Considerando BNF un estándar para la definición de reglas formales en ciencias de la computación (Haugen et al., 2003; Runde et al., 2007), asumiendo que los elementos más representativos de un diagrama de secuencia UML son los mensajes con un transmisor y receptor asociados, dado que una señal representa el contenido del mensaje, estos elementos son formalmente definidos usando BNF de la siguiente forma:

<Transmisor> → Línea de vida
 <Receptor> → Línea de vida
 <Mensaje> → (Señal, <Transmisor>, <Receptor>)

Como se sabe, Transmisor y Receptor son objetos que están representados por líneas de vida. STAIRS diferencia entre la acción de enviar y recibir un mensaje para definir formalmente un evento como se muestra en las siguientes ecuaciones.

<Transmisión> → !
 <Recepción> → ?
 <Tipo> → (Transmisión | Recepción)
 <Evento> → (<Tipo>, <Mensaje>)

Es importante que para definir formalmente los fragmentos combinados de un diagrama de secuencia hay que asumir que pueden estar vacíos, sólo un evento o uno de los fragmentos combinados claramente identificado tal que es similar a los identificados en un diagrama de secuencia. Además, cada fragmento combinado está definido recursivamente usando la regla de interacción como una secuencia de éstos incluyendo palabras claves para los fragmentos combinados (*refuse*, *assert*, *seq*, *par*, *loop set*, *alt* y *xalt*). Algunas de estas palabras son parte de UML 2.0. Los casos base de interacción ocurren cuando hay una secuencia vacía o sólo un evento. Por lo tanto, una lista de interacción está definida también recursivamente por si misma como una interacción para buscar su caso base. Una lista de interacción permite definir comportamiento potencial y obligatorio, las operaciones *alt* y *xalt* respectivamente. Las siguientes ecuaciones presentan estas definiciones formales usando BNF (Haugen et al., 2003; Runde et al., 2007).

<Vacío> → skip
 <Rechazo> → refuse [<Interacción>]
 <Afirmación> → assert[<Interacción>]
 <Secuencia Débil> → seq [<Interacción>]
 <Paralelo> → par [<Interacción>]
 <Bucle> → loop set [<Interacción>]
 <Alternativas Potenciales> → alt [<Lista de Interacción>]
 <Alternativas Obligatorias> → xalt [<Lista de Interacción>]
 <Interacción> → <Vacío> | <Evento> | <Rechazo> | <Afirmación> | <Secuencia Débil> | <Paralelo> |
 <Bucle> | <Alternativas Potenciales> | <Alternativas Obligatorias>
 <Lista de Interacción> → <Interacción> | <Lista de Interacción>

Detalle de huellas

Revisando la sintaxis de STAIRS, hay una identificación de tipos de eventos que son identificables en un diagrama de secuencia. La Figura 10 muestra un ejemplo de un diagrama de secuencia usando comportamiento potencial y obligatorio respectivamente junto a sus eventos asociados los que son identificados y descritos a continuación. La lista de eventos correspondientes asociados a la Figura 10 son:

y los correspondientes a la lista de eventos de la Figura 11 son:

Las dos últimas ecuaciones expresan la diferencia entre el comportamiento potencial y obligatorio, alt y xalt respectivamente, en la cual alt tiene dos opciones en donde cualquiera de ellas puede estar en la solución final mientras que para xalt estas dos opciones son parte de la definición, por lo tanto deben estar en la solución final.

Fig.10: Operación alt de UML / STAIRS.

Fig. 11: Operación xalt de UML / STAIRS.

Refinamiento Formal

Conociendo que el refinamiento considera tres tipos de filtros como fue previamente señalado (complemento, reducción y detalle) y el objetivo principal del refinamiento es reducir el número de comportamientos inconclusos produciendo comportamiento positivo y negativos como también asumiendo la existencia de un conjunto de comportamientos o eventos p para positivos y n para negativos respectivamente, el refinamiento está definido formalmente por:

$$(p1, n1) \rightarrow (p2, n2), \text{ si } n1 \sqsubseteq n2 \wedge p1 \sqsubseteq (p2 \cup n2)$$

La siguiente sub-sección define formalmente cada uno de los tipos de refinamiento mencionados.

Complemento

La Figura 12 muestra la idea de complemento como un tipo de refinamiento. Formalmente, STAIRS lo define como:

$$(p, n) :> (p', n') = p \sqcap p' \wedge n \sqcap n'$$

Fig. 12: STAIRS: Refinamiento por complemento.

Reducción

La Figura 13 muestra la idea de reducción como un tipo de refinamiento, STAIRS lo define formalmente como:

$$(p, n) :> (p', n') = p' \sqcap p \wedge n' = n \cup p \setminus p'$$

Fig. 13: STAIRS: Refinamiento por reducción.

Detalle

Detalle es otro tipo de refinamiento el cual significa que no se agregan directamente nuevos comportamientos. Esto es, revisando y particularizando los actuales comportamientos sin producir cambios entre los conjuntos de comportamientos existentes (Haugen et al., 2003). Un ejemplo común de detalle es presentado en (Haugen et al., 2003; Kobro et al., 2009; Haugen et al., 2005), redefiniendo los parámetros de un mensaje o cambiando su condición de sincronización. Las secciones Nuevos Operadores y Herramientas presentan la semántica y la sintaxis de los nuevos operadores de fragmentos detallados de STAIRS, xalt y refuse (Haugen et al., 2003), respectivamente, los cuales ayudan a mejorar el detalle ofreciendo un comportamiento estricto bien definido en los diagramas de secuencia.

Nuevos Operadores

STAIRS presenta nuevos operadores para eliminar interpretaciones ambiguas. Dos de ellos se describen en la siguiente lista:

Operador xalt: El operador xalt es una solución correcta para evitar interpretaciones ambiguas que están presentes en alt. Directamente alt no indica si las alternativas son obligatorias o no. La Figura 11 muestra el uso de xalt y describe las reglas asociadas de STAIRS. Como se expresa en (Kobro et al., 2009), existen situaciones en la cual los desarrolladores creen que todas las alternativas son similares y creen que es necesario implementar sólo una de ellas, produciendo situaciones erróneas cuando alt expresa un comportamiento obligatorio pero no está explícitamente indicado.

Operador refuse: El operador refuse es similar a neg de UML 2.0 porque indica comportamiento negativo con una sola gran diferencia: refuse es estricto, ya que en STAIRS un comportamiento negativo no puede cambiar a positivo. Este principio representa la base principal de STAIRS y de su lógica de refinamiento.

Herramientas

Existe una herramienta la cual sus autores llamaron Escalator (ESC, 2002), que fué desarrollada como parte de un estudio de doctorado en la Universidad de Oslo, Noruega. Sin embargo, esta herramienta es relativamente antigua ya que la última versión es del 2009. Aunque existe documentación para su instalación, usa una versión antigua de Eclipse (3.4 o 3.2) pero la versión necesaria no está claramente establecida. Uno de los autores de este ensayo intentó instalar Escalator usando ambas versiones de Eclipse sin resultados positivos. Además, intentó instalar la primera versión de esta herramienta sin obtener buenos resultados. Por lo tanto, considerando la potencia de este lenguaje de formalizar los diagramas de secuencia, al menos una herramienta debería estar disponible para soportarlo.

CONCLUSIONES

Claramente, UML es un estándar que tiene problemas intrínsecos con la ambigüedad debido a constructores que lo permiten y producen modelos no necesariamente correctos. En la búsqueda de soluciones al problema anterior aparece STAIRS que hace un par de apreciaciones las que aportan a la disminución de la ambigüedad.

A diferencia de OCL que permite formalizar diversos diagramas UML, STAIRS es un lenguaje formal para modelar diagramas interactivos de UML, especialmente diagramas de secuencia. La actual versión de UML ha extendido los diagramas de secuencia para soportar conceptos relativamente avanzados en el desarrollo de software como por ejemplo concurrencia y secciones críticas. Sin embargo, muchas de ellas están abiertas a la interpretación humana y son fuente de malas interpretaciones. STAIRS soluciona los errores típicos agregando nuevos operadores no ambiguos y es compatible con la nuevas tendencias de desarrollo de software rápido como XP o RUP (Somerville, 2004) ya que STAIRS permite aplicar una formalización gradual sobre los diagramas de secuencia. Existen operadores STAIRS muy útiles como los revisados en este artículo; xalt y refuse los cuales permiten evitar malas interpretaciones. Actualmente, hay una herramienta, Escalator, que no está actualizada y su instalación es paradójicamente ambigua considerando que soporta un lenguaje formal. Otro punto importante relacionado a STAIRS es que, en primer lugar UML es el lenguaje de modelación orientado a objetos por defecto. Aunque existen nuevas metodologías de desarrollo de software como por ejemplo DSOA, UML es adaptable a estas formas de desarrollo de software. En este sentido, existen muchos trabajos actuales aplicando y extendiendo STAIRS.

Considerando que los autores actualmente trabajan con DSOA, ellos argumentan que es necesaria una consistencia entre las fases de este ciclo de desarrollo. Revisando los trabajos actuales relacionados a DSOA y los diagramas de secuencia UML, Cristian Vidal et al. (2012) ofrece una adaptación de diagramas de secuencias UML a DSOA. Sin embargo, no existen trabajos relacionados con una extensión de STAIRS para soportar DSOA. De esta forma, una idea de trabajo futuro es la extensión o adaptación de STAIRS para DSOA. Uno de los autores está actualmente trabajando en esta idea. Por otra parte, el objetivo es desarrollar una herramienta para soportar una versión de diagramas de secuencia, STAIRS y DSOA.

REFERENCIAS

Booch, G., Rumbaugh, J., y Jacobson, I., Unified Modeling Language User Guide, Segunda Edición, Addison-Wesley Object Technology Series, Chapter 18 (2005).

Fakhroutdinov, K., Sequence Diagrams. <http://www.uml-diagrams.org/sequence-diagrams.html>. Acceso: 30 de Diciembre (2012).

Fleming, S., Kraemer, E., Stirewalt, R. E. K., Xie, S., y 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, Alemania, Mayo (2008).

Haugen, Ø., y Stølen, K., STAIRS - Steps to Analyze Interactions with Refinement Semantics, UML 2003 - The Unified Modeling Language, Modeling languages and applications, 6th International Conference, 388-402, San Francisco, CA, Estados Unidos, Octubre 20-24 (2003)..

- Haugen, Ø., Ragnhild, K. E. H., Kobro, R., y Stølen, K., STAIRS: Towards Formal Design with Sequence Diagrams, *Software & System Modeling*, Springer-Verlag (2005).
- Haugen, Ø., Husa, K. E., Runde, K. R., y Stølen, K., Why Timed Sequence Diagrams Require Three-Event Semantics, In *Proceedings of Scenarios: Models, Transformations and Tools'2003*, 1-25, Springer-Verlag (2003).
- Kobro, R., Haugen, Ø., y Haugen, K., Refining UML Interactions with Underspecification and Nondeterminism, In *Nordic Journal of Computing*, 12(2):157–188 (2005).
- Kobro, R. R., y Øyvind, J., Guidelines for Developing Sequence Diagram Specifications, Exemplified for IPTV, University of Oslo, *Elektronikk 1* (2009).
- Ragnhild, K., STAIRS —Understanding and Developing Specifications Expressed as UML Interaction Diagrams, Tesis Doctoral, Faculty of Mathematics and Natural Sciences, University of Oslo, Oslo, Noruega, (Enero, 2007).
- Runde, K. R., Haugen, Ø., y Stølen, K., Refining UML Interactions with Underspecification and Nondeterminism, Research Report 325, ISBN 82-7368-278-1, ISSN 0806-3036, University of Oslo, Oslo, Noruega, Department of Informatics (2007).
- Soeken, M., Willie, R., Kuhlmann, M., Gogolla, M., y Drechsler, R., Verifying UML/OCL Models Using Boolean Satisfiability, In *Proceeding of the Conference on Design, Automation and Test in Europe (DATE '10)*, European Design and Automation Association, 3001 Leuven, Belgium, Belgium, ACM (2010).
- Soldal, M., Operational Analysis of Sequence Diagram Specifications, Tesis Doctoral, Faculty of Mathematics and Natural Sciences, University of Oslo, Oslo, Noruega (2007).
- Somerville, I., *Software Engineering*, Addison Wesley, Séptima Edición, Capítulo 4 (2004).
- ESC, The Escalator Tool, <http://heim.ifi.uio.no/~massl/escalator/>, Acceso: 30 de Diciembre (2012).
- Vidal Silva, C., Schmal, R., Rivero, S., y Villaroel, R., Extensión del Diagrama de Secuencias UML (Lenguaje de Modelado Unificado) para el Modelado Orientado a Aspectos, *Información Tecnológica*, 23(6), 51-62 (2012).
- Xie, S., Kraemer, E., y 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).