Información Tecnológica
Vol. 30(1), 263-272 (2019)
http://dx.doi.org/10.4067/S0718-07642019000100263

# Una Propuesta de Algoritmo Spin / Promela para el Análisis y Diagnóstico de Errores en Diagramas de Secuencia UML

**Cristian L. Vidal-Silva[1, 5]*, Rodolfo H. Villarroel[2], Xaviera A. López-Cortés[3] y José M. Rubio[4]**
(1) Ingeniería Civil Informática, Escuela de Ingeniería, Campus Rodelillo, Universidad Viña del Mar, Agua Santa 7055, Viña del Mar – Chile. (e-mail: cristian.vidal@uvm.cl)
(2) Escuela de Ingeniería Informática, Facultad de Ingeniería, Pontificia Universidad Católica de Valparaíso, Avenida Brasil 2241, Valparaíso – Chile. (e-mail: rodolfo.villarroel@pucv.cl)
(3) Departamento de Computación e Informática, Facultad de Ingeniería, Universidad Católica del Maule, Avenida San Miguel 3605, Talca – Chile. (e-mail: xlopez@ucm.cl)
(4) Área Académica de Informática y Telecomunicaciones, Universidad Tecnológica de Chile INACAP, Av. Vitacura 10.151, Vitacura, Santiago – Chile. (e-mail: jrubiol@inacap.cl)
(5) Escuela de Ingeniería Informática, Facultad de Ingeniería, Ciencias y Tecnología, Universidad Bernardo O'Higgins, Avenida Viel 1497, Ruta 5 Sur, Santiago – Chile. (e-mail: cristianvidal@docente.ubo.cl)

* Autor a quien debe ser dirigida la correspondencia

*Recibido Jun. 1, 2018; Aceptado Ago. 2, 2018; Versión final Sep. 4, 2018, Publicado Feb. 2019*

---

**Resumen**

Este trabajo describe las principales características de diagramas de secuencia UML, la noción de falla o error y tolerancia a fallas, y algunos tipos de fallas comunes y sus acciones de corrección en un diagrama de secuencias UML. Así, el principal objetivo de este trabajo es proponer un algoritmo para la transformación de diagramas de secuencia UML en código Spin / Promela, una herramienta de verificación formal y de detección de errores en el chequeo de modelos para un sistema de tolerancia a fallas, y así entregar explicaciones de los pasos necesarios para ajustar y corregir los diagramas afectados. El algoritmo para transformar diagramas de secuencia UML en código Spin / Promela es útil para la detección de fallas en secuencias de mensajes. Se aplica la solución propuesta sobre un diagrama simple y general de secuencias UML para analizar su código Promela y garantizar la efectividad del chequeo de modelos sobre diagramas de secuencia UML. Además, se presentan ideas de extensión de la propuesta para el análisis de diagramas de secuencias UML con la inclusión de fragmentos combinados de iteraciones.

*Palabras clave: análisis de modelos; tolerancia a fallas; UML; Spin / Promela; diagramas de secuencia.*

# An Spin / Promela Algorithm Proposal for the Analysis and Errors Diagnosis in UML Sequence Diagrams

**Abstract**

This paper describes the main characteristics of UML sequence diagrams, the notion of failure or error and fault tolerance, and some common fault types and their correction actions in a UML sequence diagram. Thus, the main objective of this work is to propose an algorithm for the transformation of UML sequence diagrams in Spin / Promela code, a formal verification and errors detection in the model checking for a fault tolerance system, and thus to provide explanations of the necessary steps to adjust and corrections the affected diagrams. The algorithm for transforming UML sequence diagrams into Spin / Promela code is useful for the detection of faults in sequence of messages. The proposed solution is applied on a simple and general UML sequence diagram to analyze its Promela code, guarantying in this way the model checking effectiveness on UML sequence diagrams. This work also presents ideas to extend the proposal for the analysis of UML sequences diagrams which include combined fragments of iterations.

*Keywords: models analysis; fault tolerance; UML; Spin / Promela; sequence diagrams*

## INTRODUCCIÓN

Tal y como enuncian Adlemo (1994) y Sukumar (2007), la tolerancia a fallas es una característica para mejorar la confiabilidad de un sistema computacional. Según (Dubrova, 2013), la tolerancia a fallas es la propiedad que le permite a un sistema seguir funcionando correctamente en caso de falla de uno o varios de sus componentes. En un principio, la tolerancia a fallas se aplicaba principalmente en cuestiones de hardware, pero ya desde hace años también se aplica en contextos de software (Adlemo, 1994). El desarrollo de sistemas computacionales con un enfoque orientado a objetos requiere el uso de un lenguaje de modelamiento como UML. UML incluye diagramas para especificar, documentar y modelar elementos estructurales y de comportamiento de un sistema de información (Vidal et al., 2013; Visser et al., 2014; Vidal et al., 2014). Justamente, los diagramas de clase UML modelan la estructura del sistema de software (estructura de los principales componentes del sistema de software), mientras que los diagramas de secuencias y los diagramas de estado modelan el comportamiento del sistema de software (Pender, 2003). Tal y como argumenta Visser et al. (2014), técnicas y herramientas prácticas para capturar y predecir el comportamiento esperado de un sistema de información.

Para un modelamiento consistente de un sistema de información, el comportamiento de dicho sistema se modela de acuerdo a sus propiedades estructurales según sus modelos estructurales. Como parte del proceso de modelado de comportamiento de las clases de un sistema de software desarrollado con una metodología orientada a objetos, con UML permite el uso de diagramas de estados y diagramas de secuencia (Pender, 2003): Un diagrama de estados permite modelar los estados y sus transiciones de estado para un objeto individual de un sistema, y también modelar los estados y transiciones de estado del sistema como un todo. Aunque un diagrama de estados no muestra explícitamente los efectos de la interacción con otros objetos del sistema, este diagrama representa completamente el comportamiento del objeto o sistema modelado según sus estados y transiciones. Por otra parte, un diagrama de secuencia UML modela las interacciones entre diferentes objetos en un sistema por medio de mensajes para escenarios definidos. Así, los diagramas de secuencia son usables para modelar el comportamiento de los objetos participantes de casos de uso definidos, según la definición de las clases y sus componentes (atributos y métodos). Teniendo en cuenta que existen múltiples escenarios posibles, es relevante su clasificación según su relevancia para el modelamiento de aquellos escenarios de comportamiento más importantes. Como una característica adicional de los diagramas de secuencia UML, estos permiten representar interacciones algorítmicas a través de fragmentos combinados que corresponden a diagramas de secuencia secundarios dentro de un diagrama padre: una composición de comportamiento de arriba hacia abajo.

Después de considerar la idea de desarrollar un sistema de software con tolerancia a fallas, una cuestión importante es la identificación de cada falla, así como su tratamiento en etapas iniciales del proceso de desarrollo como lo son la especificación y diseño de software. En ese sentido, Spin / Promela representan una herramienta y lenguaje de programación para el chequeo de modelos (model checking en inglés) y para la verificación formal de sistemas de software y detección de fallas (Holzmann, 2004; Ben-Ari, 2008; Allen, 2008); Promela (del inglés Process or Protocol meta language) es el lenguaje de programación y Spin (del inglés Simple Promela Interpreter) es su intérprete. De hecho, como ya existen trabajos de Spin / Promela con diagramas de estados UML (Schäfer et al., 2001; Ebnenasir y Cheng, 2006; Chen y Kulkarni, 2012), y dado que los diagramas de secuencia UML permiten modelar el comportamiento algorítmico de objetos participantes en todos los escenarios de un sistema de software, el principal objetivo y contribución de estos artículos es proponer y aplicar un algoritmo para obtener el código Promela a partir de instancias de diagramas de secuencia UML, y así mediante el uso de Spin, verificar la corrección de dicho código en la detección de fallas de comunicación, interbloqueos y código no ejecutable (Ben-Ari, 2008). Además, la promoción de usar diagramas de secuencia UML sin excluir otras herramientas de modelado para el comportamiento como los diagramas de estados y colaboración UML representa otra contribución de este artículo.

## METODOLOGÍA

En un desarrollo de software orientado a objetos, después de definir actores y casos de uso de la aplicación, y luego de establecer los principales elementos estructurales del sistema de software -clases con sus atributos y métodos- debe definirse un modelo de comportamiento de alto nivel del sistema de software. Por lo tanto, es necesario definir escenarios para describir y conocer las características de comportamiento de cada elemento estructural del sistema. En este contexto, los diagramas de secuencia UML permiten describir escenarios del sistema de software, y saber cómo los objetos participantes interactúan y reaccionan bajo ciertas condiciones en esos escenarios (Pender, 2003).

Los diagramas de secuencia UML permiten modelar escenarios de interacciones entre los objetos participantes de las clases y los actores de un sistema de software (Larman, 2004). Los elementos participantes interactúan, se comunican, usan mensajes. Además, los diagramas de secuencia UML permiten

establecer fragmentos combinados para admitir concurrencia, comportamiento alternativo, ciclos y excepciones para modelar el comportamiento algorítmico de un conjunto importante de acciones de un sistema (Larman, 2004; Miles y Hamilton, 2006). La figura 1 muestra un ejemplo básico de diagrama de secuencia UML sin considerar el uso de fragmentos combinados.

Tal y como lo señalan Pender (2003) y el trabajo de Miles y Hamilton (2006), los diagramas de secuencia de UML son muy adecuados para mostrar colaboraciones entre los objetos de un sistema, pero no son tan buenos para la definición precisa del comportamiento de estos. Para modelar el comportamiento de un solo objeto para diferentes casos de uso, es conveniente usar un diagrama de estado UML. Aunque los diagramas de estados UML permiten también modelar el comportamiento de todo el sistema de software, deducir qué objetos están involucrados en cada cambio de estado del sistema aun representa un problema que no se resuelve directamente con dicho diagrama. Con los diagramas de secuencia UML es posible saber que no existe un solo escenario de ejecución para el sistema de software.

Independientemente de los beneficios de usar diagramas de secuencia UML y la utilidad del algoritmo propuesto en este artículo, es necesario considerar como base los trabajos previos con diagramas de estados UML para generar código Promela (Kulkarni, 1999; Schäfer et al., 2001; Ebnenasir y Cheng, 2006; Chen y Kulkarni, 2012) con el objetivo global de ir produciendo sistemas de software sin fallas o errores o con el menor número posible de ellos. Entonces, es recomendable modelar el comportamiento de un sistema de software con diagramas de secuencia y diagramas de estados UML. Además, si es posible validar los errores y fallas en los productos de ambas herramientas con el uso de Spin / Promela antes de las siguientes etapas en el proceso de desarrollo de un sistema de software, esto es, con generación de código Promela para la aplicación del verificador de modelo Spin para validar la corrección del sistema de software, definitivamente ambas herramientas UML son de un alto valor para la producción de software de calidad.

## CASO DE ESTUDIO

Se presenta un ejemplo simple de diagrama de secuencia UML como caso de estudio, principalmente ya que este considera o incluye los elementos básicos de dicho diagrama, esto es, objetos, y mensajes síncronos y asíncronos de comunicación. Así, este ejemplo de aplicación se puede extender a otros diagramas sin la presencia de fragmentos combinados, para el análisis y tolerancia a fallos sobre dichos diagramas. Existen numerosos ejemplos de aplicación de diagrama de secuencias UML tales como la modelación de escenarios en un sistema Adaptativo de Control de Crucero (sistema ACC) (Ebnenasir y Cheng, 2006), o en un sistema de cuentas bancarias (Vidal et al., 2013).

La figura 1 muestra un diagrama de secuencia UML para mostrar las interacciones de dos instancias de las clases A y B, donde la instancia de clase A envía un mensaje síncrono Msg1(0) a la instancia de clase B y obtiene R1 como respuesta, luego la misma instancia de clase A envía un mensaje síncrono Msg2(1) a la misma instancia de clase B y obtiene R2 como respuesta, y finalmente la instancia de clase B envía un mensaje asíncrono a la instancia de clase A. La figura 1 fue diseñada con el uso de la herramienta en línea (DiagramaSecuencia, 2018).

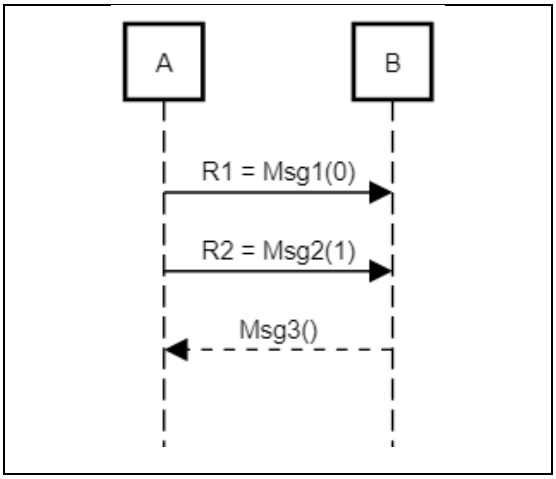


Fig. 1: Ejemplo simple de diagrama de secuencia UML.

Aun cuando la figura 1 representa un ejemplo básico de escenario para la secuencia de interacción entre objetos, tal como presenta la figura 2, dichos escenarios pueden también corresponder al detalle de interacción de los objetos involucrados a un caso de uso. Los diagramas de secuencia UML, diagramas de

estado y diagramas de colaboración permiten modelar el comportamiento de los casos de uso de sistemas de software (Miles y Hamilton, 2006). Sin embargo, la diferencia principal y la ventaja potencial de los diagramas de secuencia UML con respecto a los otros diagramas UML para modelar el comportamiento son que los primeros establecen escenarios que usualmente involucran a más de un objeto del sistema. Si bien, posiblemente haya un gran número, posiblemente infinito, de escenarios en un sistema de software, es relevante modelar situaciones críticas para describir esos escenarios con sus instancias participantes y la comunicación entre ellos, y así, después de revisar esas interacciones, poder establecer posibles estados de los objetos participantes, y entonces deducir condiciones y acciones que producen cambio de estado en dichos objetos. Por lo tanto, los diagramas de secuencia UML permiten la generación de diagramas de estados UML (Pender, 2003). Claramente, esto último representa una característica positiva del uso de diagramas de secuencia UML. Además, con la mencionada propiedad de modelar comportamiento algorítmico mediante fragmentos combinados, y la capacidad de traducirlos a código Spin / Promela para detectar y corregir fallas, definitivamente los diagramas de secuencia UML son una gran herramienta para producir modelos de software de calidad.

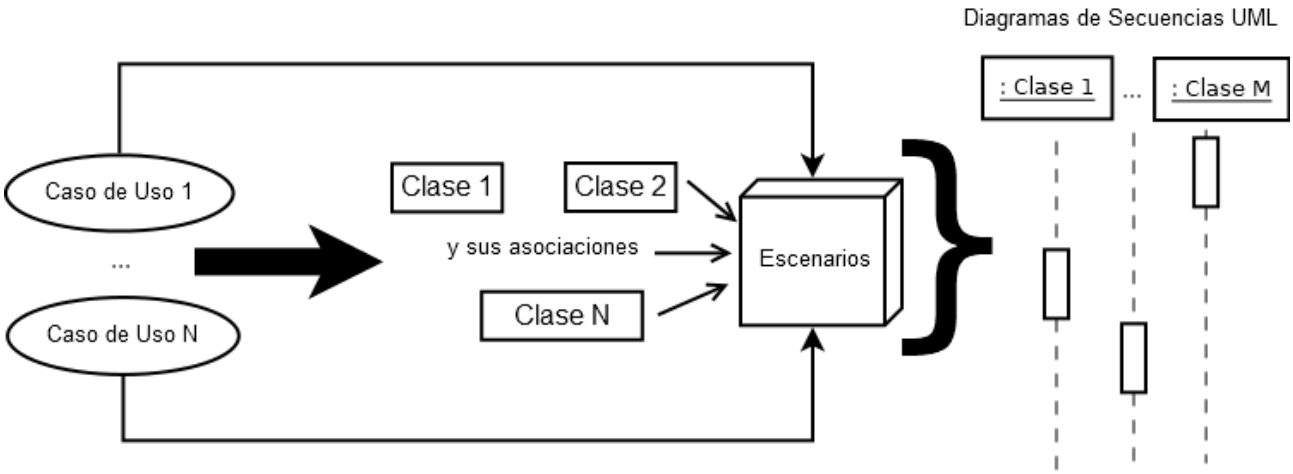


Fig. 2: Ejemplo simple de diagrama de secuencia UML.

*Tipos de Errores en Diagramas de Secuencia UML*

A continuación, se describen tipos de errores tradicionales en diagramas de secuencia UML, y también se mencionan las acciones de corrección mediante la aplicación de los principios de detector y corrector, un enfoque muy relevante para dar soporte de tolerancia a fallos en sistemas distribuidos (Adlemo, 1994):

i) Cuando se realiza una acción no permitida: considerando que las acciones básicas en un diagrama de secuencia UML son los mensajes de envío y recepción, esta falla se modela mediante la inclusión de mensajes nuevos cuando no se los espera. Esta situación puede ser viable en escenarios distribuidos en los que los objetos participantes no conocen de restricciones acerca de los mensajes aun no recibidos cuyas características no han sido informadas. Modelar esta falla claramente depende del caso de estudio porque se requieren acciones adicionales. Además, una forma de tratar esta falla es, usando una vista de tolerancia a fallas, agregar instancias de detector y corrector como se muestra en (Kulkarni, 1999) para el escenario del sistema ACC de (Ebnenasir y Cheng, 2006). Estos objetos se incluyen como objetos participantes en el diagrama de secuencia UML, y se traducen al código Spin / Promela para simular también una solución a ese problema.

ii) Cuando un mensaje se pierde por una no recepción: aunque un diagrama de secuencia UML asume que cada mensaje enviado por un objeto fuente es recibido por un objeto destino, esta situación solo suele ser cierta en sistemas no distribuidos. Sin embargo, en un sistema distribuido, esta situación no siempre es cierta, y es relevante considerar la naturaleza sincrónica y asincrónica de los mensajes. En un escenario de sistema distribuido con mensajes síncronos, un emisor espera que un mensaje de confirmación de recepción continúe su trabajo, y si el mensaje se pierde, el remitente debe volver a enviar el mensaje original. De manera similar, cuando el objetivo recibió el mensaje original, pero se perdió su mensaje de confirmación, entonces el objeto de destino debería reenviar un mensaje de confirmación de recepción. Asumiendo la presencia de corrector y detector, esta situación debe ser coordinada y decidida por esas instancias. Esta situación se simula en el código Spin / Promela con la eliminación de una instrucción de recepción en un objeto objetivo, y la eliminación de una acción de recepción de un mensaje de confirmación de recepción en un objeto fuente. La corrección relacionada incluye instancias de detector y corrector para detectar y resolver esta situación, respectivamente.

Como de costumbre, el detector envía mensajes periódicos a cada objeto en el sistema, y el objeto fuente informa sobre la no recepción de un mensaje, y el objetivo informa si recibió y envió una confirmación de recepción, o el objeto destino no recibe el original mensaje, luego el corrector y el detector deciden qué acciones se siguen para esos procesos. Es importante saber que, para simular este escenario en Spin / Promela, es necesario incluir un nuevo hilo para cada objeto del sistema que se encargue de informar que su padre está esperando la recepción de un mensaje: Mensaje perdido para un mensaje no enviado: Aunque el diagrama de secuencia UML asume que cada mensaje enviado por un objeto fuente es recibido por un objeto destino, similar al tipo de falla descrito anteriormente, esta situación es solo verdad en un sistema no distribuido.

En un sistema distribuido con mensajes síncronos, un receptor espera que un mensaje continúe su trabajo, y si el mensaje no se recepciona aún, entonces el emisor debe reenviar ese mensaje. De manera similar, cuando el objetivo recibió el mensaje enviado, pero su mensaje de confirmación aún no ha enviado, el objeto destino debe enviar su mensaje de confirmación de recepción y verificar su recepción. Asumiendo la presencia de instancias de corrector y detector, ellas deben trabajar de manera coordinada y decidida ante esta situación. Al igual que en la falla descrita anteriormente, ante la presencia de mensajes asíncronos, una solución simple es transformarlos en mensajes síncronos y resolver los mensajes perdidos para que no se envíen usando las instancias de detector y corrector.

Esta situación se simula en el código Spin / Promela con la eliminación de una instrucción enviada en un objeto fuente. De nuevo, como en la falla y corrección antes descritas, la corrección relacionada de esta falla incluye instancias de detector y corrector para la detección y resolución de esta situación, respectivamente. Como de costumbre, el detector envía mensajes periódicos a cada objeto en el sistema, y el objetivo informa de que no se recibe el mensaje original, luego el corrector y el detector deciden qué acciones se siguen para esos procesos. Es importante saber que, para simular este escenario en Spin / Promela, es necesario incluir un nuevo hilo para cada objeto del sistema que se encarga de informar que su padre está esperando la recepción de un mensaje.

## RESULTADOS Y DISCUSIÓN

Esta sección primero se presentan los elementos propios de Spin / Promela como herramienta para la detección de inconsistencias, para entonces describir el algoritmo para traducir diagramas de secuencia UML en código Spin / Promela. Esa traducción que se aplica en el caso de estudio ya descrito muestra la aplicación de principios para la detección de fallas.

### *Spin / Promela*

Tal como indican Holzmann (2004) y Ben-Ari (2008), Promela es un lenguaje para la verificación de modelos mediante la definición y creación dinámica de procesos concurrentes para el modelamiento de sistemas de comunicación en entornos distribuidos. Mediante el chequeador o revisor de modelos Spin (un intérprete de código Promela) se puede verificar si el funcionamiento del sistema modelado se ajusta o no al comportamiento deseado mediante el chequeo de la presencia de bloqueos mortales entre procesos (del inglés deadlock), recepción no especificado, la pérdida de mensajes y código que no se ejecuta en el tiempo.

La sintaxis de Promela es similar a C con la definición de un método o función principal init y de otras funciones o procesos adicionales mediante el uso de proctype. Los procesos se ejecutan mediante el comando run. La declaración de variables de tipo numéricas desde 1 a 32 bits (Holzmann, 2004) y también con el soporte de arreglos de esos tipos de datos con el soporte de variables locales, globales y como parámetros para las funciones de procesos (Holzmann, 2004). Así mismo, Promela soporta la definición de estructuras condicionales y de repetición cuya sintaxis difiere de la de C. No se entregan mayores detalles de los mismos ya que no se usan en este ejemplo, aun cuando serían útiles para la definición de fragmentos combinados tipo loop.

Promela soporta además la definición de canales de comunicación para el envío y recepción de mensajes entre procesos. Dichos canales tienen una estructura de almacenamiento tipo cola o FIFO (del inglés first input first output, esto es, primero en entrar primero en salir) y se declaran con el tipo de datos chan. Los canales de comunicación en Promela trabajan de manera síncronos o asíncronos entre procesos. Spin / Promela usa los símbolos ? y ! para enviar y recibir un valor a través de un canal de comunicación, respectivamente.

Con todo lo anterior, se puede afirmar que Promela es un lenguaje de modelamiento de procesos para la verificación de sistemas paralelos (Ben-Ari, 2008). La figura 3 ilustra un código Promela simple adaptado de (Holzmann, 2004) donde se ejecutan 3 procesos, el principal y dos instancias de proceso_ejemplo.

```
proctype proceso_ejemplo(byte x)
{
        printf("x = %d, Id Proceso = %d\n", x, _pid)
}

init {
        run proceso_ejemplo(0);
        run proceso_ejemplo(1)
}
```

Fig. 3: Ejemplo simple de diagrama de secuencia UML.

*Algoritmo para traducir diagramas de secuencia UML en código Promela*

En general, en el código Spin / Promela, cada proceso representa un participante de un diagrama de secuencia UML, esto es, un origen y/o destino de mensajes. Como se indicó anteriormente, cuando un proceso A envía un mensaje al proceso B, A solicita la ejecución de un método de B asociado con el nombre del mensaje. Usando una vista de Spin / Promela, A es el remitente y B es el receptor del mensaje, y cuando A espera por los valores del método, B genera y envía esos valores usando un canal adicional -un canal R_nombreOriginal con sus parámetros de salida.

Es absolutamente relevante tener en cuenta la naturaleza de la sincronización de cada mensaje en un diagrama de secuencias UML -sincrónico o asincrónico, ya que el tamaño de un canal en Promela permite determinar ese comportamiento (Holzmann, 2004; Ben-Ari, 2008). En Promela, un canal con un tamaño igual a 1 representa un canal asíncrono, mientras que un canal con un tamaño igual a 0 representa un canal de comunicación rendez-vous-sincrónico. Además, para indicar los parámetros de un canal, estos se definen con un parámetro de nombre simbólico de tipo mtype, para luego indicar el tipo de cada uno de los parámetros del mensaje. Según (Holzmann, 2004; Ben-Ari, 2008) mtype se utiliza para la definición de nombres simbólicos de contantes numéricas. Así, el nombre de un mensaje define el nombre de un canal de comunicación cuyos parámetros son, primero un parámetro de nombre simbólico y la lista de los tipos de parámetros del mensaje original en el diagrama de secuencia UML. Aun cuando, este modelo admite la presencia de mensajes sin parámetros, no se permite directamente el uso de parámetros tipo cadena de caracteres. Como una solución, está la definición de dicho tipo de datos como un arreglo de números (Promela soporta la definición de tipos de datos con typedef).

La figura 4 ilustra la traducción en el código Spin / Promela del ejemplo de diagrama de secuencia UML de la figura 1. Hay una situación especial cuando un participante se envía mensajes a sí mismo. La solución más simple es usar un canal asincrónico con un tamaño de búfer igual a 1 para que dicho proceso se envíe y reciba su propio mensaje. A continuación, se indican los pasos necesarios para traducir un diagrama de secuencia UML básico sin fragmentos combinados a código Promela:

1. Definir un nombre simbólico para los parámetros;

2. Identificar mensajes que devuelven valores;

3. Identificar y crear un canal de tamaño de búfer de 0 para cada mensaje síncrono. Dichoe canal tiene el mismo nombre que el mensaje, y tiene un primer parámetro mtype, y se indica a continuación el tipo de cada uno de sus parámetros originales el mismo orden después de mtype. Es importante considerar si un mensaje sincrónico devuelve valores; y cuando esto ocurre, se crea un canal R_nombreOriginal con un buffer de tamaño 0 y define sus parámetros;

4. Identificar y crear un canal con un tamaño de buffer de 1 para cada mensaje asincrónico.

Al igual que los mensajes sincrónicos, un canal asíncrono tiene el mismo nombre que el mensaje, y sus parámetros tienen su tipo original en el orden original, y un parámetro adicional mtype se incluye como el primero en considerar el uso de parámetros. Nuevamente, si un mensaje asíncrono devuelve valores, entonces se crea un canal R_nombreOriginal con un búfer de tamaño 1 y se definen sus parámetros; 5. Definir un proceso para cada participante en el diagrama de secuencia UML, y cada proceso envía y recibe mensajes en el orden establecido en el diagrama, y considerando posibles valores de retorno. Los valores de retorno se indican directamente en un mensaje en un diagrama de secuencia. 6. Definir un proceso de inicio que ejecuta cada uno ya definido.

Como ya se mencionó, la figura 2 presenta el código de Promela para el diagrama de secuencia de la figura 1 la cual no utiliza fragmentos combinados. Para mayores detalles e información adicional acerca de fragmentos combinados en los diagramas de secuencia UML, se sugiere revisar (Pender, 2003; Larman, 2004; Miles y Hamilton, 2006).

```
mtype = {Parameters}
chan Msg1   = [0] of {mtype, byte}
chan R_Msg1 = [0] of {mtype, byte}
chan Msg2   = [0] of {mtype, byte}
chan R_Msg2 = [0] of {mtype, byte}
chan Msg3   = [0] of {mtype}

proctype a()
{
 byte R1, R2;

 Msg1!Parameters(0);
 R_Msg1?Parameters(R1);

 Msg2!Parameters(1);
 R_Msg2?Parameters(R2);

 Msg3?Parameters;
}

proctype b()
{
 byte R1, R2;

 Msg1?Parameters(R1);

 ///Generating Output Value R1
 R1 = 5;
 R_Msg1!Parameters(R1);

 Msg2?Parameters(R2);
 ///Generating Output Value R1
 R2 = 10;
 R_Msg2!Parameters(R2);

 Msg3!Parameters
}

init {
   run a();
   run b();
}
```

Fig. 4: Diagrama de secuencia UML ejemplo y código Spin / Promela asociado.

*Algoritmo para obtener código Spin / Promela de un diagrama de secuencias UML con fragmento combinado loop*

Esta solución representa una extensión del algoritmo para transformar diagramas de secuencia UML en código Spin / Promela ya presentado para la traducción de diagramas de secuencia con fragmentos combinados de ciclo loop. Sin duda, siguiendo los principios de esta propuesta, es posible determinar cómo expresar en código Spin / Promela fragmentos combinados adicionales. En fragmentos combinados de ciclo loop existe una condición lógica para finalizar el mismo, esto es, si la condición es positiva, las acciones dentro del bucle se ejecutan -conjunto de mensajes entre los participantes, y después de eso la condición se evalúa nuevamente. En caso de un valor falso, el fragmento combinado de bucle loop finaliza su trabajo y se ejecuta

la siguiente acción fuera del bucle. Por lo tanto, hay procesos activos y procesos no activos en el ciclo, y en su traducción Spin / Promela, cada proceso activo tendría un ciclo con una condición lógica para continuar iterando. Además, cada proceso no activo presentaría solo un enunciado condicional, todos con una misma condición de ciclo. La figura 5 presenta un ejemplo de diagrama de secuencia con el uso del fragmento combinado loop para agregar figuras (todas las figuras disponibles) en un cuadro o plantilla de dibujo para obtener una figura, luego pintarla y finalmente devolver dicha figura pintada (Gutiérrez, 2018).

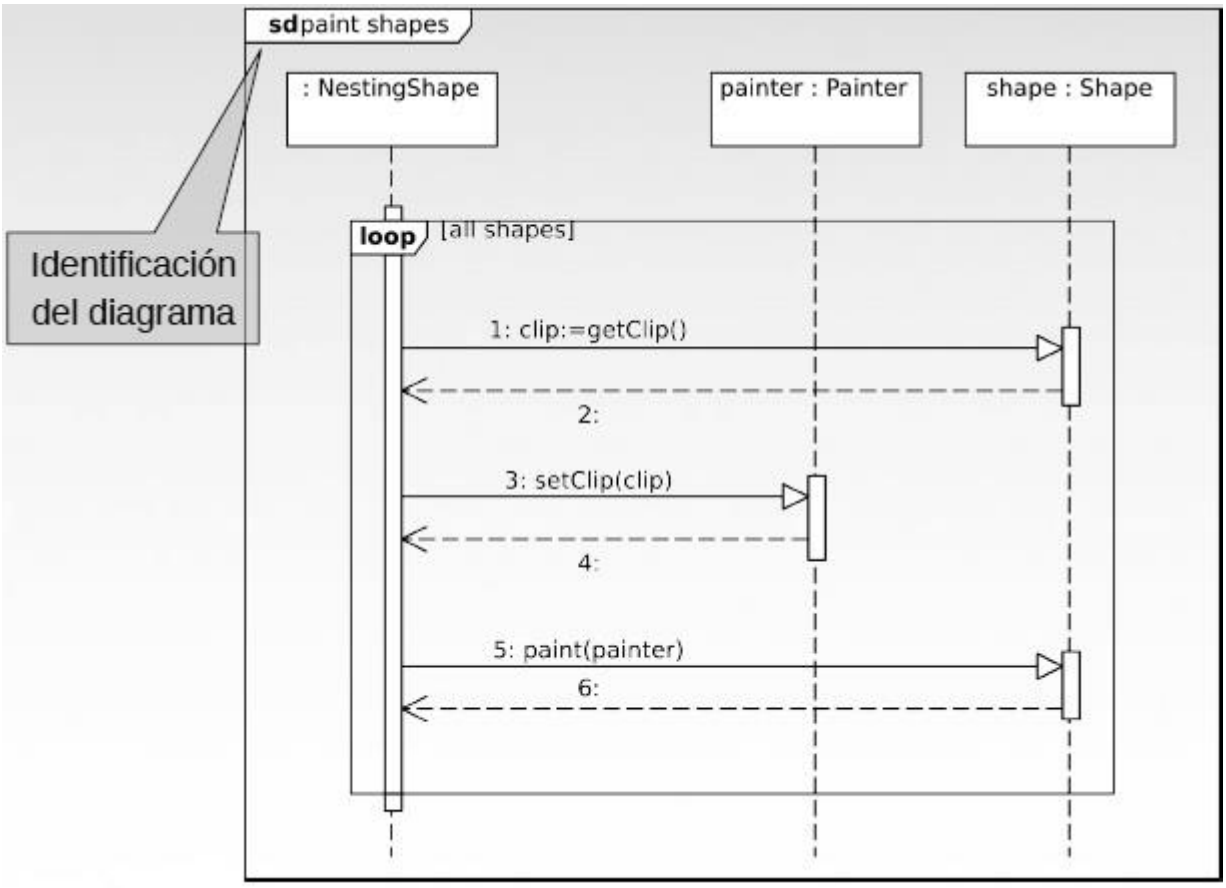


Fig. 5: Ejemplo de diagrama de secuencia UML con el uso de fragmento combinado loop.

En la práctica, debido a que la condición es la misma para los procesos activos y no activos en un fragmento combinado de ciclo loop, las variables de condición son variables globales en el código Spin / Promela asociado. Además, el valor inicial de cada variable de condición no debe generar una terminación inmediata del ciclo sin alguna iteración.

Es importante considerar que los procesos activos que envían y reciben mensajes asíncronos en un fragmento combinado de ciclo loop, debido a que un emisor asíncrono no espera por una confirmación de recepción, estos pueden proceder a estar en una iteración futura con el uso de comparación con los receptores que aún esperan por posibles mensajes perdidos en iteraciones previas. Claramente, la presencia de mensajes asíncronos puede originar fallas -pérdidas de mensajes, y la verificación de modelos revela esta situación. Además, es relevante teniendo en cuenta que los mensajes asíncronos son parte de la naturaleza de los sistemas distribuidos (Sukumar, 2007), y este tipo de sistemas de información, tales como internet y redes sociales, cada día es más parte de la vida humana.

Aunque es relevante, para diagramas de secuencia UML con el uso de fragmento combinado de ciclo loop, definir el número de iteraciones, es necesario considerar que también hay bucles con un número no preciso de iteraciones. Lograr determinar el final del ciclo en dichos escenarios no es una acción directa, por lo cual se puede generar un ciclo infinito lo cual representa otra posibilidad de falla o error.

**DISCUSIÓN FINAL**

Revisando los resultados de esta propuesta y comparándolos con los resultados de (Ebnenasir y Cheng, 2006) que generan código Spin / Promela a partir de diagramas de estados UML, es necesario destacar que ambos artículos permiten la obtención del código Spin / Promela para la validación y la corrección de los modelos. Aunque, para modelar el comportamiento, los diagramas de secuencia UML requieren más espacio

que los diagramas de estados UML, los diagramas de secuencia UML también son más expresivos que los diagramas de estados UML ya que en estos últimos, la información no es directamente deducible. Los diagramas de estado UML para un sistema de información también se pueden obtener a partir de los diagramas de secuencia UML, lo que al revés no siempre es muy viable. Definitivamente, con los diagramas de secuencia UML es posible modelar un escenario más amplio que con el uso de otras técnicas UML, aun cuando es muy necesario basarse en diagramas de estructuras del sistema información para así lograr una consistencia entre los diagramas del sistema.

Para validar la coherencia en el comportamiento de los mensajes, Spin / Promela permite definir tipos de datos estructurados, solo con atributos, no con métodos, ya que Spin / Promela no es un lenguaje orientado a objetos. Por lo tanto, el uso de Spin / Promela para un refinamiento completo no es una solución simple, y sería más fácil aplicar una herramienta adicional para validar el refinamiento de los diagramas de secuencia UML antes de su traducción al código Spin / Promela. Definitivamente, esto representa uno de los trabajos actuales de los autores de este artículo.

Como trabajo futuro, ya que este trabajo indica los pasos para traducir un diagrama de secuencia UML en código Spin / Promela, está entonces la idea de definir un algoritmo general para la generación de código Spin / Promela para su aplicación en el análisis y explicaciones de corrección sobre cualquier caso de estudio de diagramas de secuencia UML, ya que, como este trabajo presenta, Spin / Promela permite simular la interacción entre procesos para descubrir errores de ejecución. Para un refinamiento completo de los diagramas de secuencia UML, se requiere un paso previo para validar la sintaxis de los mensajes para los cuales los diagramas de clase UML deben ser una entrada adicional.

## CONCLUSIONES

De los resultados presentados, del análisis y discusión de los mismos, se obtienen las siguientes conclusiones:

1. Modelar y validar el comportamiento crítico de un sistema es de gran relevancia ya que constituye una herramienta de validación de la semántica y del funcionamiento futuro del mismo.

2. Debido a la detectar de fallas en la comunicación de los procesos de una solución Spin / Promela, la traducción de diagramas de secuencia UML al código de dichas herramientas, esto es, los participantes a procesos y los mensajes a canales de comunicación, permite el refinamiento para lograr la consistencia de dichos diagramas. Sin embargo, para una completa validación de la correcta sintaxis de los mensajes en un diagrama de secuencia UML con respecto a métodos de las clases del sistema, detalle de los componentes de las clases de los objetos participantes son absolutamente necesarios.

3. Esta propuesta establece un conjunto de pasos para traducir diagramas de secuencia UML en el código de Promela, y estos pasos se pueden implementar en una aplicación de software para generar el código Spin / Promela para así probar y finalmente garantizar la corrección de los diagramas de secuencia UML.

Claramente, este artículo demostró el uso de Spin / Promela para validar la corrección de los modelos de diagrama de secuencia UML, y esto es útil para encontrar errores, corregirlos y producir modelos de software de calidad para evitar errores en fases siguientes en el proceso de desarrollo de aplicaciones software, independiente de la metodología práctica que se use, pero con el modelamiento del comportamiento del sistema con diagramas de secuencia UML. Aunque Spin / Promela están vinculados a entornos distribuidos, este artículo demostró que es posible su adaptación a otro entorno no distribuido, esto es, para la validación y el chequeo de diagramas de secuencia UML.

## REFERENCIAS

Adlemo, A., Fault tolerance aspects in computerized systems, Actas de 7th Mediterranean Electrotechnical Conference, Antalya, Turquia, 3, 1024-1028, Abril (1994)

Allen, E., The beginning of model checking: A personal perspective, 25 Years of Model Checking, Springer-Verlag, 27- 45 (2008)

Ben-Ari, M., Principles of The Spin Model Checker, Springer-Verlag, Londres, Inglaterra, 67-124 (2008)

Chen, J. y S. Kulkarni, Application of automated revision for UML models: a case study, 13th International Conference on Distributive Computing and Networking, ICDCN, The Hong Kong Polytechnic University, Hong Kong, China, 31-45, Enero (2012)

DiagramaSecuencia, Sequence Diagram, org, https://sequencediagram.org/. Acceso: 26 de Abril de (2018)

Dubrova, E., Fault-Tolerant Design, Springer, ISBN: 978-1-4614-2112-2 (2013)

Ebnenasir, E. y B. Cheng, A framework for modeling and analyzing fault-tolerance, Technical Report MSU-CSE-06-05, Michigan State University, East-Lansing, Michigan, USA, Enero (2006)

Gutiérrez D., UML Diagramas de Secuencia, Universidad de Los Andes, Venezuela (2011)

Holzmann, G., The Spin Model Checker: Primer and Reference Manual, Addison-Wesley (2004)

Kulkarni, S., Component-based design of fault-tolerance, PhD Thesis, Ohio State University, Ohio, USA (1999)

Larman, C., Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Prentice Hall, Upper Saddle River, Nueva York, USA, 20-34 (2004)

Miles, R. y K. Hamilton, Learning UML 2.0, O'Reilly Media, Gravenstein Highway North, Sebastopol, CA, USA, 55-70 (2006)

Pender, T., UML Bible, Wiley Publishing, Indianapolis, USA, 120-167 (2003)

Schäfer, T., A. Knapp y S. Merz, Model Checking UML State Machines and Collaborations, doi:10.1016/S1571-0661(04)00262-2, Electronic Notes in Theoretical Computer Science, 55 (3), 357-369, Octubre (2001)

Setyautami, M., R. Hähnle, R. Muschevici y A. Azurat, A UML profile for delta-oriented programming to support software product line engineering, doi: 10.1145/2934466.2934479, Actas de la 20th International System and Software Product Line Conference, Beijing, China, 45 – 49 (2016)

Sukumar, G., Distributed System: An Algorithm Approach, Taylor and Francis Group, LLC, 1-10 (2007)

Vidal, C., L. López, S. Rivero y R. Meza, Extensión de Diagramas de Secuencia UML para el Modelamiento Orientado a Aspectos, Información Tecnológica, 24 (5), 3-12 (2013)

Vidal, C., S. Rivero, L. López y C. Pereira, Propuesta y Aplicación de Diagramas de Clases UML JPI, Información Tecnológica, 25 (5), 113-120 (2014)

Vidal, C., D. Benavides y otros tres autores, Mixing of Join Point Interfaces and Feature-Oriented Programming for Modular Software Product Line, doi:10.4108/eai.3-12-2015.2262534, Actas de 9th EAI International Conference on Bio-Inspired Information and Communications Technologies, Nueva York, USA, 433 – 437 (2016)

Vidal, C., Exploring Efficient Analysis Alternatives on Feature Models, doi:10.1145/3109729.3109747, Actas de 21st International Systems and Software Product Line Conference SPLC 2017, Sevilla, España, 150 – 155 (2017)

Visser, W., N. Bjørner y N. Shankar, Software Engineering and Automated Deduction, doi: 10.11345/2593882.2593899, Actas de On the Future of Software Engieering, FOSE 2014, ACM, India, 155 – 166 (2014) |