Revista Educación en Ingeniería
ISSN 1900-8260
Julio a Diciembre de 2012, Vol. 7, N°. 14, pp. 82-91 • © 2012 ACOFI • http://www.educacioneningenieria.org
Recibido: 17/09/2012 • Aprobado: 20/11/2012 • Versión final: 06/12/2012

UNA GUÍA GENERAL PARA LA ESPECIFICACIÓN Y VERIFICACIÓN FORMAL DE REQUERIMIENTOS USANDO EVENT-B™ Y RODIN™

TOWARDS A GUIDELINE FOR FORMAL SPECIFICATION AND VERIFICATION OF REQUIREMENTS USING EVENT-B™ AND RODIN™

Holmes Giovanny Salazar Osorio, Harvin Jessid Rengifo Romero, Liliana Esther Machuca Villegas, Jesús Alexander Aranda Bueno,
Universidad del Valle, Cali (Colombia)

Resumen

El modelamiento formal es una técnica utilizada comúnmente para especificar y verificar modelos matemáticos de un sistema de *software* a través de métodos formales. *Theorem Proving* hace parte de dichos métodos, el cual utiliza la lógica de primer orden y la teoría de conjuntos para verificar modelos matemáticos.

Para llevar a cabo el proceso de especificación de un sistema existe una gran variedad de lenguajes, uno de ellos es *Event-B*™ y su respectivo entorno de desarrollo *Rodin*™, ambos proveen los componentes necesarios para realizar el modelamiento y la verificación formal de sistemas. Este artículo presenta una guía para llevar a cabo el proceso de especificación y verificación formal de requerimientos basadas en el modelamiento formal, de esta manera se explora la posibilidad ofrecida por *Event-B*™ y *Rodin*™, como herramientas de apoyo al proceso de verificación.

Palabras clave: modelamiento formal, métodos formales, *Theorem Proving*, *Event-B*™, *Rodin*™, programación e ingeniería de software

Abstract

Formal modeling is a technique commonly used to specify and verify mathematical models of software systems through formal methods; Theorem Proving is part of such methods, which uses the first-order logic and set theory to verify mathematical models.

To perform the specification process of a system there are several languages, one of them is Event-B™ and their respective development environment Rodin™, both provides the necessary components for the

modeling and formal verification of systems. This paper presents some guidelines for carrying out the process of formal specification and verification of requirements by exploring the use of Event-B™ and Rodin™, as tools to support the verification process.

Keywords: formal modeling, formal methods, Theorem Proving, Event-B™, Rodin™, programming and software engineering

1. Introducción

Los sistemas de software son cada vez más complejos, por lo que las exigencias sobre su desarrollo son mayores. Es necesario que el desarrollo de software sea más riguroso para obtener un producto de alta calidad. Esto busca reducir el número de errores presentes en su construcción. La identificación y tratamiento de estos errores en etapas tempranas del proyecto de *software* es crucial para disminuir los costos de su operación y evitar errores en etapas posteriores del desarrollo.

Una de las principales etapas en el desarrollo de *software* es la especificación de requerimientos. Para lograr un producto de *software* de alta calidad, los requerimientos deben cumplir con ciertas propiedades (Wiegers, 2003). Sin embargo, en algunas ocasiones los requerimientos no satisfacen dichas propiedades, lo que se traduce en errores y en mala calidad del producto que se está desarrollando. Para evitar este tipo de situaciones existen métodos formales como *Theorem Proving* (Bibel, 1987) y *Model Checking* (Clarke, 1997) que pueden ser utilizados para la verificación formal de requerimientos (Duque, 2000), (Montoya, 2010).

La verificación formal fuerza el análisis detallado de los requerimientos, que puede revelar errores potenciales que podrían de otra forma no ser descubiertos sino hasta que el sistema esté en producción. La verificación formal juega un papel importante en el desarrollo de sistemas de *software*. La especificación formal del sistema reduce significativamente el número de fallos durante la ejecución del mismo. El uso de métodos formales va en aumento a medida que los clientes lo solicitan y a medida que los ingenieros están más familiarizados con estas técnicas.

Este artículo se concentrará en la verificación formal por medio de lenguajes basados en *Theorem Proving,* que tienen como objetivo verificar la correctitud de un sistema a desarrollar a través del modelamiento y el razonamiento formal apoyado en la lógica y la teoría de conjuntos. El modelamiento formal de un sistema en estos lenguajes se lleva a cabo mediante la construcción gradual de modelos cada vez más aproximados al sistema deseado (Abrial, 2010).

Actualmente existen varios lenguajes de especificación para *Theorem Proving* como lo son *Event-B*™ (Abrial, 2010), *Isabelle* (Nipkow et al., 2002), *Agda* (Bove et al., 2009), VDM (Jones, 1990) y PVS (Owre et al., 1992); en este artículo se hablará de *Event-B*™ y su respectivo entorno de desarrollo *Rodin*™™ (Abrial, et al., 2010), éstos cuentan con una gran aceptación en la comunidad científica y han sido utilizados en grandes proyectos industriales (Badeau & Amelot 2005), (Behm et al., 1999).

Este artículo se enfoca principalmente en proponer una guía a seguir que permita explorar la viabilidad de los lenguajes basados en *Theorem Proving* como apoyo al proceso de verificación de requerimientos. En la literatura se pueden encontrar diferentes trabajos que aplican métodos formales en el modelamiento formal de diversos problemas (Bloem et al., 2005), (Cavada et al., 2009), (Cimatti et al., 1997), (Gervasi & Zowghi 2005), sin embargo el estudio de metodologías para emplear métodos formales y en concreto *Theorem Proving* es escaso (Aagaard & Leeser 1993), (Padidar 2010).

En (Aagaard & Leeser 1993) se propone una metodología basada en *Theorem Proving* para verificar formalmente programas funcionales en particular. A diferencia de (Aagaard & Leeser 1993), la guía propuesta en este artículo se puede aplicar a cualquier tipo de sistemas modelables en *Event-B*™, esto incluye sistemas concurrentes como el sistema de tráfico de autos que se presentará en la sección IV.

En (Padidar 2010) se hace un estudio para identificar los desafíos que enfrenta un usuario novato al emplear

demostradores de teoremas, su estrategia consiste en modelar un caso de estudio extenso en Event-B™ y registrar cada acción llevada a cabo para completar el respectivo caso. Por otra parte, este artículo se orienta hacia una metodología para demostradores de teoremas que puede ser usada tanto por usuarios novatos como por expertos.

El documento se encuentra estructurado de la siguiente forma: en la sección 2, se presentan las características y propiedades de *Event-B*™ y *Rodin*™ con el fin de realizar una contextualización de los conceptos relacionados con la notación utilizada en las pautas propuestas; en la sección 3, se proponen la guía a seguir basada en el modelamiento formal en *Event-B*™ para la verificación formal de requerimientos; en la sección 4, se presenta un caso de estudio en el cual se aplica la guía propuesta, en éste se modela un sistema basado en el tráfico de autos teniendo en cuenta los requerimientos del mismo; y para finalizar, en la sección 5 se encuentran las conclusiones de este trabajo.

2. Características y propiedades de Event-B™ y Rodin™

Event-B™

Los métodos formales son un conjunto de tendencias de desarrollo de *software* en donde la especificación, verificación y diseño de componentes se realiza utilizando bases sólidas como notaciones, lenguajes, herramientas y técnicas basadas en teorías de alto fundamento matemático. *Event-B*™ es un método formal para el análisis y modelamiento de sistemas, el cual utiliza la teoría de conjuntos y lógica de primer orden como notación de dicho modelamiento.

La especificación o modelo de un sistema en *Event-B*™ se compone de dos partes: la parte estática y la parte dinámica. La parte estática contiene: los tipos de datos que se representan mediante conjuntos, las *constantes* del sistema y los axiomas, que definen las propiedades de las constantes y conjuntos; la parte estática también es llamada el contexto del modelo. La parte dinámica -también llamada máquina- contiene: las variables presentes en el modelo, los invariantes que expresan las propiedades de las variables y los eventos, los cuales describen acciones que ocurren en el sistema y modifican el valor de las variables; los eventos a su vez se dividen en dos partes, en la primera se ubican las guardas, que son condicionales con los que el sistema debe cumplir antes de que se realice la acción; en la segunda se especifican las acciones con las cuales se modifica el valor de una variable (Abrial, 2010), (Rueda, 2012).

Generalmente, cuando se piensa en modelar un sistema es de vital importancia iniciar con un modelo básico e ir incrementando las características del mismo. *Event-B*™ tiene como particularidad el uso de refinamientos, lo que facilita la representación de sistemas en distintos niveles de abstracción en la medida en que se introducen gradualmente detalles y complejidad al modelo. Esto se realiza con el fin de acercar el modelamiento a la realidad, alcanzando de esta forma un alto grado de precisión del modelo. Cuando ocurre el refinamiento de un modelo, los contextos y máquinas (y sus componentes) anteriores al refinamiento pasan a ser llamados abstractos y los contextos y máquinas (y sus componentes) posteriores al refinamiento pasan a ser llamados concretos (Abrial, 2010).

Anexo a estas funcionalidades, *Event-B*™ hace uso de pruebas matemáticas para verificar la consistencia entre los diferentes niveles de refinamiento, lo que garantiza que cada nivel de abstracción generado cumpla con las propiedades verificadas en los grados de abstracción de más alto nivel (Deploy Project, 2012). Estas pruebas se conocen con el nombre de pruebas de obligación (Rueda, 2012) y se utilizan para determinar si la especificación es consistente.

Rodin™

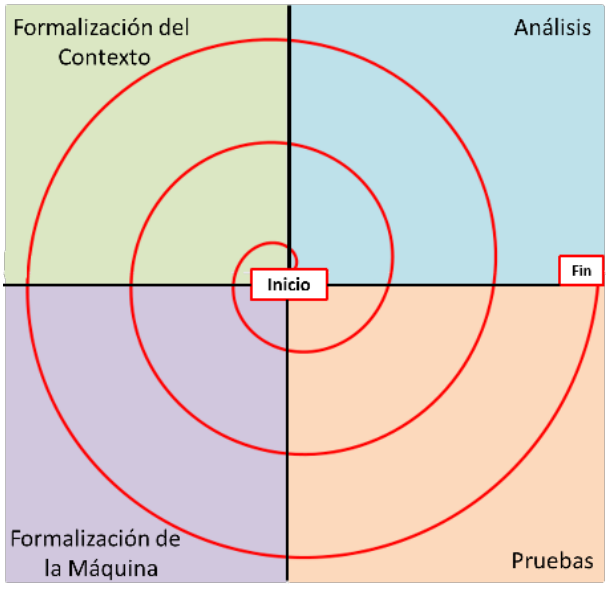
Es una herramienta que provee soporte efectivo para la construcción y verificación de modelos en el lenguaje *Event-B*™ (Deploy Project, 2012). Es una plataforma *Open Source* basada en el entorno de desarrollo Eclipse distribuida bajo la versión 1.0 de la licencia CPL (Free Software Foundation, 2012). La principal razón por la cual *Rodin*™ hace parte del *framework* de Eclipse se debe a que permite que sea más extensible mediante complementos o *plugins*, los cuales son elementos que facilitan la integración de la plataforma con distintas funcionalidades tales como: probadores, editores, generadores automáticos de código, administradores de requerimientos, entre otros (Deploy Project, 2012a).

Rodin™ posee dos entornos o perspectivas: La perspectiva de *Event-B*™ y la perspectiva de Pruebas. La perspectiva de *Event-B*™ dispone de una serie de elementos que facilitan la edición del modelo en la plataforma, utilizando las operaciones y conceptos propios de *Event-B*™. La perspectiva de pruebas es un entorno en el cual *Rodin*™ brinda apoyo al proceso de verificación formal de requerimientos. Comúnmente, la verificación de las pruebas se realiza de manera automática, pero en el caso de que las pruebas de obligación no sean efectuadas automáticamente, *Rodin*™ proporciona los mecanismos necesarios para realizar la verificación de manera interactiva.

3. Guía para la especificación y verificación formal de requerimientos usando *Event-B™*

En esta sección se plantean unas orientaciones para llevar a cabo el proceso de especificación y verificación formal de requerimientos partiendo de su especificación en lenguaje natural hasta su representación en *Event-B*™. Se plantea un proceso compuesto de 4 fases: análisis, formalización del contexto, formalización de la máquina y pruebas. Estas fases siguen un modelo en espiral, con el fin de aprovechar el enfoque basado en refinamientos del modelado en *Event-B*™. En la Figura 1 se puede ver el esquema general que representa el proceso.

Figura 1. Esquema General del proceso adoptado para la verificación formal de requerimientos



A continuación se definen cada una de las fases que hacen parte de la guía para la verificación formal de requerimientos apoyada en *Event-B*™:

Análisis: en la fase de análisis, la primera acción a realizar es elegir la estrategia de diseño, que consiste en analizar todos los requerimientos y seleccionar aquellos que son más adecuados para especificar el primer modelo. Se recomienda empezar con los requerimientos que describan las características básicas del sistema. Una vez realizada la selección de requerimientos iniciales se debe escoger cuál o cuáles requerimientos se agregarán a los siguientes modelos para constituir los diferentes refinamientos. Esta acción no es necesaria en aquellos casos en que el sistema a modelar no sea muy complejo. Después de seleccionar la estrategia de diseño se procede a identificar y separar los requerimientos dinámicos de los estáticos, los primeros describen comportamientos, acciones o formas en las que interactúan los elementos del sistema (requerimientos dinámicos) por su parte los segundos describen características o propiedades de los elementos del sistema (requerimientos estáticos).

Formalización del contexto: Después de identificar los requerimientos estáticos, se debe crear una constante por cada propiedad de un elemento descrito en estos requerimientos. En caso de que exista una propiedad que pueda contener múltiples valores, además de crear una constante por cada valor, se debe crear un conjunto y luego definir un axioma en el cual relacione por medio de un operador matemático el conjunto con cada uno de esos valores. Una vez definidas todas las constantes se definirá el tipo de cada una de éstas, para esto se crea un axioma por cada constante, en el cual se especificará el tipo. Un tipo de constante puede ser un conjunto o un tipo de número (por tipo de número se entiende el conjunto de números naturales, reales, enteros, entre otros). *Por último, cualquier otra propiedad de una constante que se deba asumir como una verdad absoluta en el resto del modelo debe ser especificada por medio de un axioma.*

Formalización de la m*áquina*: Luego de formalizar el contexto, el siguiente paso es definir la máquina del modelo con base en los requerimientos dinámicos, para esto es recomendable realizar las siguientes acciones:

- Por cada elemento descrito en los requerimientos se crea una variable.
- Asignar el tipo de variable por medio de los invariantes. El tipo puede ser un conjunto definido en el contexto o un tipo de número. Además del tipo de variable, se deben declarar invariantes cuando: Existe algún tipo de relación entre una o más variables y una o más constantes, existe algún tipo de relación entre dos o más variables, existen propiedades en el sistema con las cuales una variable siempre debe cumplir.
- Después de declarar las variables e invariantes, se deben crear los eventos de inicialización, estos eventos indicarán el valor inicial que toma cada una de las variables antes de comenzar el sistema.
- Por último, se declaran los eventos a partir de aquellos requerimientos que reflejan acciones o formas en las que interactúan los elementos del sistema.

Para los refinamientos se debe tener en cuenta lo siguiente: si hay eventos que necesiten modificarse, se crea una nueva máquina (máquina concreta) para refinar los eventos existentes en la máquina original (máquina abstracta). Si en la máquina concreta se crean nuevas variables (variables concretas) y estas tienen relación con las variables de la máquina abstracta (variables abstractas) se debe especificar dicha relación por medio de invariantes. Si en la máquina concreta existen eventos que involucren variables abstractas que tengan relación con variables concretas deben modificarse de tal forma que estos sólo involucren las variables concretas.

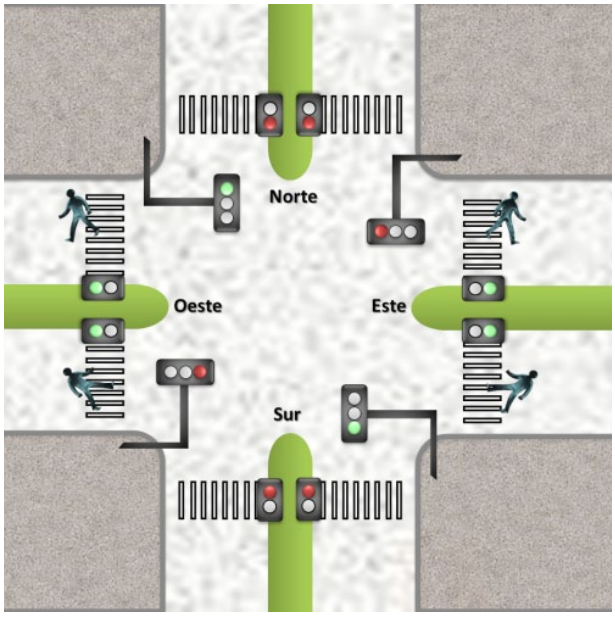
Pruebas: a cada máquina se le deben realizar pruebas con el objetivo de determinar si la especificación es consistente. Es decir, se debe verificar que cada evento cumple con las propiedades fundamentales del sistema, las cuales son expresadas a través de los axiomas e invariantes. Para cumplir con este objetivo en Rodin™ se realizan 3 tipos de pruebas (Rueda, 2012): de factibilidad, de consistencia y de ausencia de bloqueos. Las dos primeras se encargan de verificar que las variables, sus inicializaciones y modificaciones sean en todo momento consistentes con los invariantes y axiomas del modelo. Por su parte, las pruebas de ausencia de bloqueos se encargan de comprobar que el sistema no incurra en puntos muertos, esto puede ocurrir cuando las guardas o condicionales de dos eventos son contradictorias.

4. Caso de estudio: sistema de tráfico de autos

A continuación se aplica el proceso descrito en la sección anterior para la especificación y verificación de propiedades de un sistema de control de tráfico de autos, esta especificación se basa en los ejemplos que describen (Deploy Project, 2012) y (Robinson, 2011). Pese a que este sistema ha sido previamente estudiado, en este trabajo se presenta con el fin de exponer cómo esta propuesta puede utilizarse para su implementación. En este sistema interactúan cuatro semáforos vehiculares y ocho semáforos peatonales que están ubicados en distintas posiciones: Norte, Este, Sur y Oeste, los cuales son los encargados de preservar de manera segura y oportuna la movilidad de los autos y peatones que transitan exclusivamente en los sentidos: Norte-Sur, Sur-Norte, Este-Oeste y Oeste-Este (ver Figura. 2).

Cada semáforo vehicular posee un comportamiento que es definido por luces de color rojo, verde y amarillo. Por su parte, los semáforos peatonales poseen un comportamiento definido por luces de color rojo y verde. Los semáforos (vehiculares y peatonales) que se encuentran ubicados en las posiciones Norte y Sur poseen un comportamiento idéntico, así como también los semáforos ubicados en las posiciones Este y Oeste.

Figura 2. Sistema de control de tráfico de autos



Requerimientos: a continuación se precisa el comportamiento que el sistema de control de tráfico debe seguir

1. Cada semáforo vehicular debe tener 3 tipos de luces: roja, amarilla y verde.
2. Cada semáforo peatonal debe tener 2 tipos de luces: roja y Verde.
3. La secuencia para el cambio de luz de los semáforos vehiculares debe ser la siguiente: de roja a verde, de verde a amarilla y de amarilla a roja.
4. El sistema no debe permitir que los semáforos vehiculares ubicados en la vía Norte-Sur estén en luz verde o amarilla al mismo tiempo que los semáforos vehiculares ubicados en la vía Este-Oeste.
5. El sistema debe permitir que los semáforos peatonales cambien a luz verde siempre y cuando los semáforos vehiculares estén en luz roja.
6. El sistema debe permitir que los semáforos vehiculares que estén ubicados sobre una misma vía (Norte-Sur o Este-Oeste) tengan el mismo comportamiento.
7. El sistema debe permitir que los semáforos peatonales que estén ubicados sobre una misma vía (Norte-Sur o Este-Oeste) tengan el mismo comportamiento.
8. Los semáforos vehiculares ubicados en la vía Norte-Sur solo pueden estar en luz amarilla cuando los semáforos vehiculares ubicados en la vía Este-Oeste estén en luz roja y viceversa.

Especificación: a continuación se detalla el proceso de especificación del modelo de Event-B™ según las fases descritas en la sección anterior:

1. Primera iteración:

a. Análisis: la primera acción a realizar es definir la estrategia de diseño, primero se diseñará un modelo que describa el comportamiento de los semáforos vehiculares, esto es, incluir los requerimientos 1, 3, 4, 6 y 8; posteriormente se refinará este modelo de tal manera que involucre el comportamiento de los semáforos peatonales, es decir, incluir los requerimientos 2, 5 y 7.

Una vez seleccionada la estrategia de diseño, se eligen los requerimientos estáticos y dinámicos; como se estableció en la sección anterior, los requerimientos estáticos son aquellos que definen propiedades de los elementos u objetos existentes en el sistema. Para este primer modelo, el requerimiento 1 define las propiedades de un semáforo; por esta razón este requerimiento es estático. Debido a que en los requerimientos restantes sólo definen el comportamiento de los semáforos vehiculares en diferentes situaciones, se establecen los requerimientos 3, 4, 6 y 8 como dinámicos.

b. Formalización del contexto: en la fase anterior, el requerimiento 1 fue seleccionado como estático, éste describe que cada semáforo vehicular debe tener 3 tipos de luces: roja, amarilla y verde. Se pueden tomar las luces como una característica del semáforo que puede tomar 3 valores distintos (rojo, amarillo o verde), entonces en el contexto se definirán 3 constantes, una para cada color y un conjunto al cual se declarará como *LUCES*. Posteriormente por medio del axioma 1 (*axm1*) se define el tipo y la relación existente entre cada una de las constantes:

SETS
LUCES
CONSTANTS
Rojo
Amarillo
Verde
AXIOMS
axm1: partition (LUCES, {Rojo}, {Amarillo}, {Verde})

c. Formalización de la máquina: una vez definido el contexto, se pasa a formalizar la parte dinámica de nuestro modelo. Aquí se especifica el comportamiento de los semáforos, el cual está descrito por los requerimientos 3, 4, 6 y 8.

Primero se declaran 2 variables (*semáforosNS* y *semáforosEO*), las cuales describirán el comportamiento de los semáforos vehiculares ubicados en la vía Norte-Sur y en la vía Este-Oeste. Después, por medio de las invariantes 1 y 2 (*inv1*, *inv2*) se especifican que ambas variables son efectivamente semáforos vehiculares:

inv1: semáforosNS $\in$ LUCES
inv2: semáforosEO $\in$ LUCES

Las variables semáforosNS y semáforosEO junto con sus respectivas invariantes describen el requerimiento 6. Luego con los invariantes 3 y 4, se cumple con el requerimiento 4:

inv3: semáforosNS $\Longrightarrow$ {Verde, Amarillo} SemáforosEO = Rojo
inv4: semáforosEO $\Longrightarrow$ {Verde, Amarillo} semáforosNS = Rojo

Luego de haber declarado las variables e invariantes del modelo, se debe definir los eventos de inicialización y los eventos que describirán las diferentes secuencias existentes para el cambio de luces en los semáforos. Para efectos prácticos sólo se detallará la secuencia de cambio de luces para el caso en el que los semáforos están ubicados en la vía Norte-Sur:

Evento **Inicialización**:
semáforosNS := Rojo
semáforosEO := Rojo
Evento **Pasar_Verde_NS**:
WHEN
grd1 : semáforosNS = Rojo
grd2 : semáforosEO = Rojo
THEN
act1 : semáforosNS := Verde
Evento **Pasar_Amarillo_NS**:
WHEN
grd1 : semáforosNS = Verde
grd2 : semáforosEO = Rojo
THEN
act1 : semáforosNS := Amarillo
Evento **Pasar_Rojo_NS**:
WHEN
grd1 : semáforosNS = Amarillo
THEN
act1 : semáforosNS := Rojo

El evento *Pasar_Verde_NS* tiene como condicional que ambos semáforos deben estar en luz roja antes de cambiar la variable semáforosNS a luz verde, esto cumple con lo descrito por el requerimiento 4. En el evento *Pasar_Amarillo_NS*, se puede observar que antes de cambiar la luz de los semáforos Norte-Sur a Amarillo, la luz de estos mismos semáforos debe estar en Verde y la luz de los semáforos ubicados en la vía Este-Oeste debe estar en Rojo. Esta situación concuerda con lo dicho en el requerimiento 8.

d. Pruebas: una vez implementado el modelo, *Rodin*™ por medio de sus probadores automáticos realiza las pruebas de obligación necesarias para probar que el sistema es consistente, es decir verificar que cada parte de la especificación cumple con las propiedades fundamentales del modelo, las cuales fueron descritas por los requerimientos 1, 3, 4, 6 y 8. Las pruebas realizadas por *Rodin*™ para este sistema fueron de factibilidad y consistencia, las cuales lograron verificar que no había ninguna inconsistencia en el modelo. Esto quiere decir que se logró comprobar que los eventos no presentan conflictos con los axiomas e invariantes existentes en el modelo.

2. Segunda Iteración:

En esta iteración se incorporará al sistema el comportamiento de los semáforos peatonales, es decir, se consideran los requerimientos 2, 5 y 7.

a. Análisis: puesto que ya fue definida una estrategia de diseño, en esta iteración no es necesario definir una nueva. Por lo tanto, se incluye al modelo existente las características definidas en los requerimientos 2, 5 y 7, los cuales describen el comportamiento de los semáforos peatonales. La primera acción a realizar en esta etapa será definir los requerimientos estáticos y dinámicos. El requerimiento 2 describe las propiedades de los semáforos peatonales, por esta razón se clasifica este requerimiento como estático. Los requerimientos 5 y 7 son clasificados como dinámicos, pues describen el comportamiento de los semáforos peatonales.

b. Formalización del contexto: como se puede observar en la descripción del requerimiento 2, las luces que poseen los semáforos peatonales ya están definidas en el contexto del modelo anterior, por esta razón no es necesario realizar un nuevo contexto.

c. Formalización de la máquina: la primera acción a realizar es declarar 2 variables (*sem_peatonalNS* y *sem_peatonalEO*), las cuales describirán el comportamiento de los semáforos peatonales ubicados en la vía Norte-Sur y en la vía Este-Oeste. Después, con los invariantes 1 y 2 (*inv1, inv2*) se especifica que ambas variables son semáforos peatonales:

inv1 : sem_peatonalNS $\in$ {Verde, Rojo}
inv2 : sem_peatonalEO $\in$ {Verde, Rojo}

Las variables *sem_peatonalNS* y *sem_peatonalEO* junto con sus respectivos invariantes describen los requerimientos 2 y 7. A través de las invariantes 3 y 4, se expresa el requerimiento 5, además se establece la relación entre las variables del modelo anterior (variables abstractas) y las variables del nuevo modelo (variables concretas):

inv3 : semáforosNS $\Longrightarrow$ {Verde, Amarillo} sem_peatonalNS = Rojo
inv4 : semáforosEO $\Longrightarrow$ {Verde, Amarillo} sem_peatonalEO = Rojo

Antes de pasar a refinar los eventos de la máquina abstracta se crean las variables que se utilizarán en esos eventos y se establece por medio de invariantes (*inv5*, *inv6*) su relación con las variables abstractas:

inv5 : semNS = semáforosNS
inv6 : semEO = semáforosEO

Como se puede observar en las invariantes 5 y 6 (*inv5*, *inv6*) las variables concretas (*semNS, semEO*) tendrán el mismo comportamiento en los eventos refinados que las variables abstractas (*semáforosNS, semáforosEO*).
La última acción es definir los eventos de inicialización y los eventos concretos, que no son más que los eventos de la máquina abstracta refinados. De la misma manera en que se hizo la iteración anterior, todas las variables que representan a los semáforos vehiculares y peatonales serán inicializadas en Rojo. Para definir el comportamiento de los semáforos peatonales en esta iteración se deben refinar todos los eventos de la primera iteración, es decir se deben refinar todos los eventos abstractos. A continuación se ilustrará la manera en que se refinó uno de los eventos de la primera iteración:

Evento **Pasar_Rojo_NS1**:
REFINES
Pasar_Rojo_NS
WHEN
grd1: semNS = Amarillo
grd2: sem_peatonalNS = Rojo
THEN
act1: semNS := Rojo
act2 : sem_peatonalNS := Verde

Como se puede ver hay un nuevo componente en el evento y es la sección *REFINES*, en esta sección se hace referencia al evento abstracto que se está refinando. En este evento se agrega una guarda y una acción, dichos elementos describen el proceso de cambio de luz de rojo a verde y también evitan que los semáforos peatonales cambien a luz verde cuando los semáforos vehiculares están en luz roja, situación que concuerda con el requerimiento 5.

d. Pruebas: terminado de implementar el sistema, *Rodin*™ se realizan una vez más las pruebas de obligación necesarias para probar que el sistema es consistente. En esta iteración se realiza un nuevo tipo de pruebas de obligación además de las pruebas de factibilidad y consistencia. Este nuevo tipo de pruebas son llamadas de ausencia de bloqueos, las cuales lograron verificar que las guardas de los eventos abstractos son preservadas por las guardas de los eventos concretos y no presentan contradicciones entre ellas, esto se hace con el fin de evitar que el sistema entre en un punto muerto.

Si se tiene en cuenta que todos los axiomas e invariantes, así como todos los eventos fueron diseñados conforme las características estipuladas en los requerimientos y que además *Rodin*™ no encontró inconsistencia en ellos, se infiere que dichos requerimientos han sido ***verificados formalmente.***

5. Conclusiones

A través de este trabajo se han llegado a las siguientes conclusiones:

- La verificación formal de requerimientos por medio de herramientas basadas en probadores automáticos (*Theorem Proving*) como *Rodin*™ es posible. El éxito de estas herramientas depende del nivel de descripción del sistema a modelar. A partir de una especificación precisa de las características que debe tener el sistema y el modelo del sistema, los resultados que ofrecen estas herramientas serán confiables y se podrá determinar, con un alto grado de precisión, que el modelo es consistente (o inconsistente) con respecto a la especificación de requerimientos.

- El uso de métodos formales en la etapa de análisis del ciclo de vida del *software* aumenta el tiempo de desarrollo, por otra parte garantiza un ahorro en tiempo en la etapa de producción, ya que facilita la detección de errores en etapas tempranas haciendo más confiable el software que se está desarrollando. En la medida que se necesite un *software* altamente confiable, robusto, seguro los métodos formales se convierten en una herramienta útil de apoyo para su desarrollo.
- El nivel de experiencia del usuario en cuanto a fundamentos matemáticos relacionados con la teoría de conjuntos y lógica de primer orden, resulta ser un factor importante al momento de modelar sistemas utilizando herramientas como *Event-B*™ y *Rodin*™.
- La guía propuesta se presenta como un instrumento útil para la verificación formal de requerimientos, ya que proporciona un conjunto de actividades estructuradas que facilitan el modelamiento formal de sistemas en *Event-B*™, además proporciona la ayuda necesaria para traducir los requerimientos de un sistema a un modelo a partir de su especificación original.
- La guía presentada en este artículo se puede usar como base para la creación de una metodología para la verificación formal de requerimientos.

Referencias

Aagaard, M., Leeser, M. (1993). *A Theorem Proving Based Methodology for Software Verification*. Recuperado en noviembre de 2012 de http://ecommons.cornell.edu/bitstream/1813/6101/1/93-1335.pdf.

Abrial, J. R. (2010). *Modeling in Event-B - System and Software Engineering*. Cambridge University Press.

Abrial, J., Butler, M., Hallerstede, S., Hoang, T., Mehta, F., & Voisin, L. (2010). Rodin: an open toolset for modeling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (STTT), 12* (6), 447–466.

Badeau, F., & Amelot, A. (2005). Using B as a high level programming language in an industrial project: Roissy VAL. *ZB 2005: Formal Specification and Development in Z and B,* 15 - 25.

Behm, P., Benoit, P., Faivre, A., & Meynadier, J. M. (1999). METEOR: A successful application of B in a large project. *FM'99—Formal Methods,* 712 - 712.

Bibel, W. (1987). *Automated Theorem Proving Artificial Intelligence*. (2da. Ed.). Braunschweig: Vieweg & Sohn.

Bloem, R. Cavada, R. Pill, I. Roveri, M. & Tchaltsev, A. (2005). RAT: A tool for the formal analysis of requirements. *In Proc. 19th CAV, LNCS* 4590, 263–267.

Bove, A., Dybjer, P. & Norell, U. (2009). A Brief Overview of Agda - A Functional Language with Dependent Types. Theorem Proving in Higher Order Logics , 5674 (LNCS), 73-78.

Cavada, R. Cimatti, A. Mariotti, A. Mattarei, C. & Micheli, A. (2009). Supporting Requirements Validation: The EuRailCheck Tool. *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering,* 665 – 667.

Cimatti, A. Giunchiglia, A. Mongardi, G. Romano, D. Torielli, F. Traverso, P. (1997). Model Checking Safety Critical Software with SPIN: An Application to a Railway Interlocking System. *Proceedings.Third SPIN Workshop, R. Langerak, ed., Twente Univ,* 1-13.

Clarke, E. (1997). Model checking. *Foundations of software technology and theoretical computer science, 54–56.*

Deploy Project. (2012). *Rodin User's Handbook*. Recuperado en junio de 2012 de http://handbook.Event-B.org/current/html/index.html.

Deploy Project. (2012a) *Rodin Plug-ins*. Recuperado en junio de 2012 de http://wiki.Event-B.org/index.php/Rodin_Plug-ins.

Duque, J. G. (2000). *Especificación, verificación y mantenimiento de requisitos funcionales con técnicas de descripción formal.* PhD thesis, Departamento de Tecnología de las Comunicaciones. University of Vigo.

Free Software Foundation. (2012). Common Public License: Licencia Pública Común. Recuperado en junio de 2012 en http://www.gnu.org/licenses/license-list.es.html#SoftwareLicenses.

Gervasi, A. & Zowghi, A. (2005). Reasoning about inconsistencies in natural language requirements. *ACM Transactions on software engineering and methodology*, 277-330.

Jones, C. B. (1990). Systematic software development using VDM, Vol. 2. Prentice Hall.

Montoya, E. S. (2010). Métodos formales e Ingeniería de Software. *Revista Virtual Universidad Católica del Norte,* (30)*,* 1–26.

Nipkow, T., Paulson, L. C., & Wenzel, M. (2002). Isabelle/HOL: a proof assistant for higher-order logic, *Springer Verlag, 2283.*

Owre, S., Rushby, J., & Shankar, N. (1992). PVS: A prototype verification system, *Automated Deduction—CADE-11,* 748–752.

Padidar, S. (2010). A Study In The Use Of Event-B For System Development From A Software Engineering Viewpoint. MSc Dissertation, University of Edinburgh. Recuperado en noviembre de 2012 de http://www.ai4fm.org/papers/MSc-Padidar.pdf

Robinson, K. (2011). *System Modeling & Design Using Event-B*. The University of New South Wales. Recuperado en agosto de 2012 de http://wiki.event-b.org/images/SM%26D-KAR.pdf

Rueda, C. (2012). *Desarrollo Formal de Software: Obligaciones de Prueba*. Recuperado en febrero de 2012 de http://cic.javerianacali.edu.co/wiki/lib/exe/fetch.php?media=materias:desarrollo:clase4_5-obligaciones.pdf.

Wiegers, K. (2003). *Software Requirements*. (2da. Ed.). Microsoft Press.

Sobre los autores

Holmes Giovanny Salazar Osorio
Estudiante de décimo semestre de Ingeniería de Sistemas.Universidad del Valle, Tulúa (Colombia)
holmes.salazar@correounivalle.edu.co

Harvin Jessid Rengifo Romero
Estudiante de décimo semestre de Ingeniería de Sistemas. Universidad del Valle, Tulúa (Colombia)
harssed1126@gmail.com

Liliana Esther Machuca Villegas
Ingeniera de Siatemas con Maestría en Ingeneiría con énfasis en Ingeniería de sistemas y computación. Integrante del Grupo de Investigación CAMALEÓN. Profesora de la Escuela de Ingeniería de Sistemas y Computación de la Universidad del Valle, Cali (Colombia)
liliana.machuca@correounivalle.edu.co

Jesús Alexander Aranda Bueno
Ingeniero de Siatemas con Doctorado en Ingeniería con énfasis en Ingeniería de sistemas y computación y con Doctorado en Informática. Integrante del Grupo de Investigación AVISPA. Profesor de la Escuela de Ingeniería de Sistemas y Computación de la Universidad del Valle, Cali (Colombia)
jesus.aranda@correounivalle.edu.co