Especificación Formal de Requerimientos de un Sistema de Control de Paso de Vehículos Pesados en Puentes

Cuarite Silva Cesar, Estudiante[1], Valdivia Baldarrago Christian, Estudiante[1], y Vera Churata Suling, Estudiante[1]
Vidal Duarte Elizabeth, Mentora[1]
[1]*Universidad Nacional de San Agustín,Perú, ccuarite@unsa.edu.pe, svera@unsa.edu.pe*
cvaldivia@unsa.edu.pe, *evidald@unsa.edu.pe*

Abstract– Existen riesgos críticos en el recorrido de vehículos pesados a través de puentes que pueden verse afectados o derrumbarse a consecuencia de pesos excesivos en sus plataformas. En la ciudad de Arequipa, una de las principales ciudades mineras del Perú, se presenta el problema que actualmente no existe un mecanismo de control para garantizar el control de vehículos de carga pesada. Ante este problema se propone realizar un Sistema de Control del paso de vehículos a través de puentes. En este artículo presentamos la primera fase de este proyecto. Dado lo crítico del sistema ya que el factor de riesgo es alto no sólo en infraestructura sino en vidas humanas se hace uso de especificaciones formales para la especificación de requerimientos. El aporte de nuestro trabajo se basa en que al ser un modelo formal este puede ser reutilizado en otras minas en donde se necesite implementar un sistema de control de vehículos de carga pesada. Así mismo nos permite mostrar el gran aporte que es el uso de una notación matemática dentro del proceso de desarrollo de software como herramienta para reducir la ambigüedad de los requerimientos.

Keywords-- Especificación formal, requerimientos, VDM++

I. INTRODUCCIÓN

Actualmente, en algunos países emiten permisos especiales a aquellos camiones que excedan el límite de peso de la jurisdicción de la carretera [1]. Esto porque existen factores de riesgo en los puentes, ya que pueden sufrir deterioros importantes o colapsos a causa de pesos excesivos en sus respectivas plataformas. Es así un sistema de control del paso de vehículos a través del puente se convierte en una necesidad crítica más aún si los puentes quedan cerca a centros poblados que implicaría riesgo en daños físicos y vidas humanas.

Los métodos formales permiten el uso de notaciones matemáticas que ayudan en el proceso de implementación de sistemas críticos [7], reduciendo las potenciales ambigüedades que ocurren en la interpretación de los modelos gráficos tradicionales. Este trabajo hace uso de VMD++ como lenguaje de especificación formal para el Sistema Propuesto.

El principal aporte del presente trabajo es mostrar que el uso de una notación matemática dentro del proceso de desarrollo de software ayuda a incrementar la confiabilidad en la especificación de requerimientos.

El resto del artículo está organizado de la siguiente manera: En la sección 2 se presentan trabajos relacionados al presente trabajo, en los cuales se presentan algunas aplicaciones de especificaciones formales con diferentes enfoques y algunos proyectos que utilizaron técnicas de especificación formal. En la sección 3 se describe los fundamentos en los que se basa el presente trabajo y las características de VDM++ para el desarrollo del proyecto agregando una breve descripción de la herramienta utilizada en nuestra propuesta. En la sección 4, se presenta el caso de aplicación, con una breve descripción del problema y describiendo los requerimientos del sistema propuesto. Además, se muestra el desarrollo de las especificaciones formales de las clases de un modelo de solución para el problema descrito y la validación de las especificaciones utilizando la herramienta VDM++ ToolBox. Finalmente presentamos nuestras conclusiones en la sección 5.

II. TRABAJOS RELACIONADOS

Una de las propuestas desarrolladas acerca de sobrecargas vehiculares [1] se centra en la creación de modelos de confiabilidad para evaluar la seguridad estructural de los puentes de carreteras y factores propuestos de carga permitida para la verificación de sobrecarga. Además, Minguillón [2] en su trabajo desarrolla un modelo integral de carga de tráfico para la verificación de seguridad de un puente, este modelo incluye un algoritmo para la simulación de un flujo continuo de tráfico y otro para la extrapolación de los resultados, incluye también el tratamiento estadístico de las variables más significativas involucradas. Chen[3] presenta la modelación de un sistema de superestructura de puente sometido a cargas de tráfico en movimiento, la verificación del modelo se realiza de manera gráfica mediante la herramienta ADINA [4]. Además, es posible el uso de herramientas alternativas de especificación y verificación como presentan Queille y Sifakis [5] mediante un ejemplo ilustrativo del uso de la herramienta CESAR [6]. Esta herramienta permite la validación progresiva de la descripción algorítmica de un sistema de comunicación de procesos secuenciales basados en un conjunto de especificaciones de sistema.

Nuestra propuesta a diferencia de las mencionadas se basa en un modelo de verificación constante pesando los vehículos que ingresan a un puente y controlando el tráfico en el puente con semáforos. Debido a factores externos, como el clima y tiempo, puede disminuir el peso de tolerancia del puente, es por ello que nuestro modelo planteado es constante.

III. ESPECIFICACIONES FORMALES

J. Rushby [7] define a las especificaciones formales como descripciones de los diferentes sistemas computacionales utilizando notación basada en lógica formal. Las especificaciones afirman supuestos acerca del contexto en el cual el sistema opera. Las propiedades a especificar en muchos casos son demasiado grandes por lo que la herramienta a utilizar debe ser lo suficientemente resistente en el modelado de muchas propiedades.

La construcción de un software o sistema demanda un modelado, análisis y construcción, para realizar las actividades anteriores de manera eficaz y sin errores, podemos utilizar especificaciones formales las cuales son técnicas matemáticas que ayudan en la detección temprana de errores y defectos del sistema. Además, al utilizar especificaciones formales se brinda mayor fiabilidad y solidez en la etapa del diseño del software [15]. Uno de los lenguajes de especificación formal utilizado en la industria es VDM++.

A. VDM++

VDM++ [10] utiliza especificaciones formales con notación matemática para expresar con precisión los requerimientos previstos de un sistema. Dichas especificaciones se construyen con VDM++ y son llamados modelos los cuales constan de en base a clases con variables y operaciones. Las operaciones tienen como elementos principales las pre-condiciones y las post-condiciones que son un elemento clave al momento de evaluar el requerimiento previsto en el modelo [11].

VDMToolBox es una herramienta de modelado que es propiedad y desarrollada por CSK Systems [16]. En este trabajo se utiliza VDM++ como lenguaje de especificación formal mediante la herramienta VDM++ ToolBox [8

B. Estructura de Clases o modelos

Para empezar con los componentes de la estructura de las clases, En la línea 1 de la Figura 1 se muestra el primer componente el cual es el nombre de la clase. El segundo elemento de la estructura de las clases son las variables de instancia, como se muestra en la línea 5 y 6.

Como tercer elemento de la estructura de clases tenemos a las operaciones, las cuales constan de un nombre, parámetros, retorno, pre y pos condiciones y un valor de retorno. En la línea 13 se encuentra la palabra parámetro allí se define el tipo de parámetro o el tipo de variable que servirá como parámetro. En la misma línea 13 se encuentra la palabra Retorno, este espacio se define el tipo de dato de retorno del método. En las líneas 14 y 15 se definen las precondiciones y postcondiciones que se deberán cumplir para evaluar la veracidad de cada requerimiento.

Por último en la línea 18 de la Figura 1 se estable el final de la clase creada con la palabra reservada “end” seguida del nombre de la clase.

```
1 Class nombre (Nombre de la clase )
2
3 Instance variables (Definición de variables de
4                          instancia)
5       variable 1
6       variable 2
7
8       inv (Invariantes)
9
10 Operations  (Definición de los métodos a utilizar
11                  en la clase)
12
13      public nombreMetodo :Parámetro =>Retorno
14              pre  (pre condiciones)
15              pos (pos condiciones)
16              return valor-retornado;
17
18 end nombre Clase
```

Figura 1: Estructura de un modelo en VDM++

C. Tipos de datos VDM ++

Según Fitzgerald [18] los modelos en VDM++ están centrados en la definición de tipos de datos creados a partir de tipos bases abstractos, constructores de tipos, tipos de unión, registros, conjuntos, secuencias y mapeos. Además a las definiciones de estos tipos se le puede colocar una fórmula booleana invariable para denotar una propiedad que todos los elementos de ese tipo deben cumplir o respetar.

Para la validación futura de nuestro sistema necesitamos crear variables que cambian su valor en variantes de tiempo, en VDM++ pueden declarar tipos de datos como en lenguajes de programación tradicionales, con un nombre apropiado[16].

D. *Herramienta*

Para modelar nuestro sistema se utilizó la herramienta VDM++ToolBox, la cual permite desarrollar y analizar modelos precisos de sistemas informáticos[8].

Una vez realizado el modelo del sistema en Rational Rose, la herramienta permite la conversión en modelos de VDM++ para solo agregar la precondiciones, postcondiciones e invariantes en el modelo generado. La herramienta también nos permite la generación de código en un lenguaje de alto nivel.

E. *Verificación y Validación*

La verificación y validación son procedimientos independientes que se utilizan en conjunto para verificar que un producto, servicio o sistema cumpla con los requisitos y especificaciones y que cumpla con su propósito [12].

En gestión de proyectos de software es el proceso de verificar que un sistema de software cumpla con las especificaciones y su propósito, también es utilizado con frecuencia en calidad de software [13].

IV. CASO DE APLICACIÓN

En esta sección se presenta la descripción del problema con respecto al paso de vehículos a través de puentes y algunas situaciones de riesgo que podrían ser evitadas con la construcción del modelo de validación, el cual también es descrito en está sección. Además se presenta los requerimientos y un análisis parcial del sistema de control de paso vehicular a ser validado.

A. *Descripción del problema*

La importancia de los puentes en el desarrollo y en las relaciones humanas ha sido el objetivo principal del impulso para el conocimiento en la construcción y mantención de dichas estructuras.[14]

En el Perú existe una irregularidad con la calidad de materiales usados en la construcción de infraestructura, como es el caso de los puentes. Por ello, aquellos puentes construidos para el pase de vehículos pesados se debilitan ante cualquier exceso de peso y de esta forma el puente resiste cada vez menos, al exceder su peso podría hacer colapsar un puente y poner en riesgo las vidas humanas que se encuentren cerca.

Figura 2. Camiones atravesando al mismo tiempo un puente

La construcción de un software para regular el paso de estos vehículos es una alternativa importante pero muy riesgosa si no se realiza una evaluación rigurosa en el análisis de creación de dicho software. En Figura 2 se muestra un ejemplo de infraestructura de puente sobre el cual se percibe el paso de vehículos pesados, que a pesar de la seguridad que aporta el puente, pueden existir factores de riesgo excepcionales.

B. *Propuesta*

Este sistema de control se encarga de la regulación del paso de vehículos pesado en puentes. El sistema contará con un número de semáforos y mecanismos de paso sincronizados en ambos lados de los puentes. A continuación se presentan el lenguaje natural los principales requerimientos.

R1: El sistema debe gestionar un adecuado tránsito de vehículos para evitar accidentes y dañar la infraestructura del puente.

R2. Se necesitan de dos balanzas y dos semáforos, un semáforo y una balanza en cada carril del puente.

R3: El peso de cada vehículo que ingresará al puente se calculará con balanzas en los extremos del puente.

R4: El sistema hará una verificación de peso de cada uno de los vehículos que ingresarán a un determinado carril del puente.

R5: El sistema sumará el peso de los vehículos ingresados de ambas direcciones y lo comparará con el peso límite que puede soportar el puente.

R6: Para decrementar el peso total, se pesará vehículos salientes del puente con balanzas que se encontrarán en los extremos del puente.

R7: Existirá dos semáforos para retener a los vehículos y puedan ser pesados con mayor precisión.

R8: Los semáforos darán acceso a lo vehículos de cruzar el puente cuando el sistema lo indique.

C. *Diagrama de Clases*

En Figura 3 se presenta el diagrama de clases del sistema propuesto.

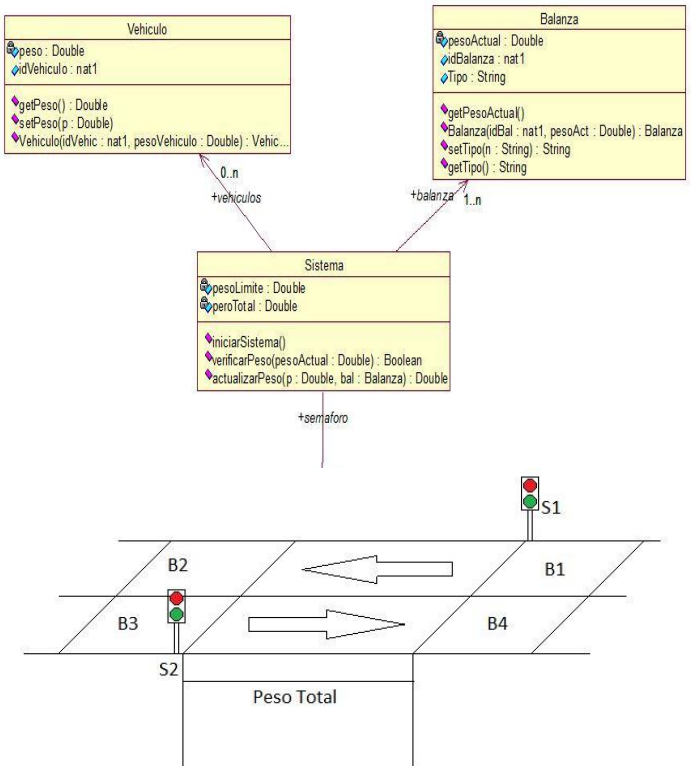


Figura 3. Diagrama de Clases del sistema de control

Como una de las soluciones a la validación del sistema, se propone el uso de VDM++ como lenguaje de especificación y modelado para el sistema de control de paso vehicular en puentes. El sistema de control es orientado a objetos y posee comportamiento en tiempo real, por lo cual la herramienta VDM++ es de mucha ayuda, ya que se puede verificar una a una las clases o entidades pertenecientes al sistema de control.

C. *Sistema de Control*

En esta subsección se presenta la descripción básica del sistema como solución propuesta para el problema descrito. Se ha considerado el uso de dos semáforos que deben ser ubicados en cada lado del puente, los cuales recibirán señales del sistema central para permitir o restringir el paso de vehículos pesados.

Adicionalmente se necesitan cuatro balanzas ubicadas de la siguiente manera: dos en la parte inicial del puente que obtendrán el peso de cada vehículo que ingresará al puente (lado de ingreso) y que saldrá del puente (lado opuesto) y de la misma forma se debe colocar dos balanzas en la parte final del puente que cumplirán la misma función de las balanzas ubicadas en la parte inicial del puente. En Figura 2 se puede ver el esquema básico deseado de los componentes que se van a utilizar para el funcionamiento del sistema.

La lógica del sistema consiste en que cada Balanza deberá de manejar una variable en común llamada “peso Total” la cual es una variable general de la clase Sistema y cambiará de valor dependiendo del peso obtenido por cada balanza, si es una balanza de entrada o salida el valor de “peso Total” será incrementado o disminuido respectivamente. Además dependiendo de “peso Total”, el sistema enviará señales de cambio a cada semáforo dependiendo del valor de “peso Total”.

Por ejemplo, si el peso límite del puente es de cien toneladas y el peso acumulado en el sistema se acerca a dicho límite en un margen de 5 toneladas, el sistema enviará señales a los semáforos para impedir el paso de algún otro vehículo pesado y el semáforo volverá a permitir el paso de los vehículos cuando el peso acumulado del sistema disminuya considerablemente a causa de que cada vehículo que sale del puente es pesado y se actualiza la variable “peso Total” y de esta forma se controla el paso de vehículos evitando riesgos de posibles colapsos o accidentes en los puentes.

Figura 4: Esquema de Componentes del Sistema

En Figura 4 se puede observar la descripción gráfica de la ubicación de cada componente del sistema. “B1”, B3” son las Balanzas ubicadas al inicio de cada carril del puente y se encargan de pesar los vehículos que ingresan al puente. “B2”y “B4”se ubican al final de cada carril para incrementar el peso que sale del puente. Dependiendo de la variable “Peso Total” se cambiará el estado de cada semáforo y controlar el paso de vehículos.

D. *Modelo en VDM++*

Se modela la case Balanza para emular el comportamiento de las balanzas físicas que se ubican en los extremos del puente.

```
1   class Balanza
2   types
3   public String=seq of char;
4   instance variables
5     public Tipo : String;
6       public idBalanza : nat1;
7     private pesoActual : real;
8     inv pesoActual>=0
9   operations
10    public getTipo : () ==> String
11    getTipo() ==
12  (return Tipo)
13    pre len Tipo >0;
14
15    public setTipo : String ==> ()
16    setTipo(n) ==
17      (Tipo:=n)
18      pre len n >0
19  post Tipo = n;
20
21    public getPesoActual : () ==> real
22    getPesoActual() ==
23      (return pesoActual);
24
25    public Balanza : nat1 * real ==> Balanza
26    Balanza(idBal, pesoAct) ==
27      (   idBalanza:=idBal;
28  pesoActual:=pesoAct)
29  pre idBal > 0 and idBal < 5;
30  end Balanza
```

Figura 5: Clase Balanza VDM++

En Figura 5, podemos observar las líneas de código de las clase “Balanza” en el cual las líneas 4-7 se definen las variables de instancia que representan el peso obtenido, el tipo de balanza y el identificador de cada balanza. El tipo de Balanza posee un tipo que no es nativo, en este caso, se define el tipo de dato “String” que representa una secuencia de caracteres, tal como se muestra en la línea 2.

En la línea 7 se define la invariante peso Actual, para indicar que el peso no debe ser negativo y finalmente la clase define operaciones a partir de la línea 9 para modificar y obtener las variables de instancia en la clase Balanza.

En Figura 6 se aprecia el código de la clase “Semaforo”. La clase semáforo es el modelo de los semáforos en físico que están en cada lado del puente controlando el ingreso de vehículos. La clase Semaforo está compuesta por dos atributos; estado definido en la línea 4, y el identificador de Semáforo definido en la línea 5.

```
1   class Semaforo
2
3   instance variables
4     private estado : bool;
5     public idSemaforo : nat1;
6
7   operations
8     public Semaforo : nat1 * bool ==> Semaforo
9     Semaforo(idSem, est) ==
10      (estado:=est; idSemaforo:=idSem);
11
12    public getEstado : () ==> bool
13      getEstado() ==
14        (return estado;);
15
16    public setEstado : bool ==> bool
17      setEstado(est) ==
18       (estado:=est; return estado;);
19
20    public cambiarEstado : () ==> bool
21    cambiarEstado() ==
22      if (estado <> false)
23      then (estado := false; return estado;)
24      else (return estado;);
25
26  end Semaforo
```

Figura 6: Clase Semaforo VDM++

El constructor de las líneas 8-10 recibe dos parámetros que es el identificador del semáforo y su estado de tipo bool (true para permitir el paso de vehículo y false para restringir el paso de vehículos).

La operación “getEstado” en las líneas 12-14 retorna el estado actual del semáforo. En la operación “setEstado“ en las líneas 16-18 recibe como parámetro una variable de tipo bool que es asignado al atributo estado. En la operación “cambiarEstado” en las líneas 20-14 modifica el valor bool del atributo estado dependiendo en cual se encuentra, si esta como “true”, lo modifica a “false” y viceversa.

La clase Vehiculo modela los datos de un vehículo real que ingresará al puente, para poder manejar variables básicas como un identificador ficticio de un vehículo y un peso asociado. En Figura 7 se puede ver que la clase Vehiculo tiene atributos: identificador y peso porque es necesario asociar los datos de un vehículo que ingresa al puente al vehículo que está saliendo del puente posteriormente y de esta forma saber que un vehículo en particular ya esta terminando su recorrido a través del puente.

```
1   class Vehiculo
2
3   instance variables
4     private peso : real;
5     public idVehiculo : nat1;
6     inv peso >= 0;
7
8   operations
9     public getPeso : () ==> real
10    getPeso() ==
11      (return peso);
12
13    public setPeso : real ==> ()
14    setPeso(p) ==
15      (peso:=p);
16
17    public Vehiculo : nat1 * real ==> Vehiculo
18    Vehiculo(idVehic, pesoVehiculo) ==
19      (peso:=pesoVehiculo;
20  idVehiculo:=idVehic);
21  end Vehiculo
```

Figura 7: Clase Vehiculo VDM++

En Figura 7, en la línea 6 del código se define la invariante para evitar que el peso que se ingresa de un respectivo vehículo sea negativo o igual a cero a partir de la línea 8 se define las operaciones de acceso y modificación de las variables de instancia en la clase Vehiculo.

La clase sistema, representa el comportamiento clave del sistema, se encarga de manipular los datos de otros objetos, inicializar los objetos (componentes) del sistema y agregar o disminuir los valores de la variable global del sistema "Peso Total" que será el atributo fundamental para el control del paso de vehículos a través del puente.

En la Figura 8 se muestra el desarrollo de la clase Sistema que está compuesta de dos atributos: "peso Total" contiene el peso acumulado de las clases vehiculos creadas, "peso Limite" contiene el valor real de peso que puede resistir la infraestructura del puente sin dañarlo. Además, a partir de la línea 9 del código se puede ver las operaciones definidas para la clase, en el cual tenemos "verficarPeso" que calcula el peso total al adicionar el peso de un nuevo Vehículo, En caso el peso total más el nuevo exceda el peso Límite. En la operación actualizar peso hace un llamado a la operación verificar peso para recién adicionar el peso del nuevo vehículo entrante

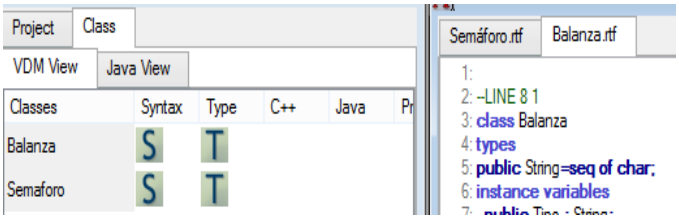
```
1   class Sistema
2
3   instance variables
4     private pesoTotal : real;
5     private pesoLimite : real;
6     public sema: seq of Semaforo := [new Semaforo(1,true), new Semaforo(2,true)];
7     public balan: seq of Balanzas:= [new Balanza(),new Balanza(), new Balanza(),new Balanza()];
8     inv pesoLimite = 70000;
9     inv pesoTotal < pesoLimite;
10
11  operations
12    public verificarPeso : real ==> Boolean
13    verificarPeso(nuevo) ==
14    if (pesoTotal+nuevo < pesoLimite)
15    then return true
16    else return false
17    pre pesoTotal > 0;
18
19    public iniciarSistema : () ==> ()
20    iniciarSistema() ==
21  is not yet specified;
22
23
24
25    public actualizarPeso : real ==> real
26    actualizarPeso(nuevo) ==
27      if (verificarPeso())
28    then pesoActual:=pesoActual+nuevo
29
30    pre pesoActual < pesoLimite and pesoActual > 100;
31
32
33  end Sistema
34
```

Figura 8: Clase Sistema en VDM++

.

E. Verificación de la sintaxis y correctitud de los tipos de acuerdo a las reglas de VDM++

En esta subsección se presenta el proceso de refinamiento del código que se produce de forma iterativa para llegar al correcto sintaxis y tipo de cada clase desarrollada. Se realiza tantas veces como sean necesarias para que la herramienta VDM Tool Box ++ no muestre ningún error de sintaxis ni de tipo. En la figura 9 se puede apreciar que la especificación se realizó de forma correcta, después de corregir errores indicados en anteriores evaluaciones. En la Figura 9 se puede



ver la validación para las clases Balanza y Semáforo, de esta forma se puede continuar el proceso de validación con la generación del código en lenguaje Java.

Figura 9: Validación de Sintaxis y Tipo de Especificación.

F. Generación de Código Java

Cuando ya se ha validado la especificación, es posible generar un código en lenguaje de programación, en este caso, código en lenguaje Java. Para el Sistema descrito, se ha realizado la generación automática de código de programación. El código obtenido de la clase Semaforo se presenta en la figura 10. Para cada clase VDM++ se genera una clase Java con el mismo nombre, de esta forma se generará elementos correspondientes para cada miembro o atributo de la clase VDM++.

```
1  import jp.co.csk.vdm.toolbox.VDM.*;
2  import java.util.*;
3  public class Semaforo implements EvaluatePP {
4    static UTIL.VDMCompare vdmComp = new UTIL.VDMCompare();
5    private volatile Boolean estado = null;
6    public volatile Integer idSemaforo = null;
7    volatile Sentinel sentinel;
8    class SemaforoSentinel extends Sentinel {
9
10     public final int Semaforo = 0;
11
12     public final int getEstado = 1;
13
14     public final int setEstado = 2;
15
16     public final int cambiarEstado = 3;
17
18     public final int nr_functions = 4;
19
20     public SemaforoSentinel () throws CGException {}
21
22
23     public SemaforoSentinel (EvaluatePP instance) throws CGException {
24       init(nr_functions, instance);
25     }
26   }
27 ;
28   public Boolean evaluatePP (int fnr) throws CGException {
29     return new Boolean(true);
30   }
31   public void setSentinel () {
32     try {
33       sentinel = new SemaforoSentinel(this);
34     }
35     catch (CGException e) {
36       System.out.println(e.getMessage());
37     }
38   }
```

Figura 10: Porción de código Java generado de la clase Semaforo

En la Figura 10 se puede apreciar el lenguaje Java generado, el cuerpo de los métodos generados en la clase respetan las especificaciones de precondiciones y poscondiciones implementados en VDM++, así como poseer el mismo nombre de operaciones y de atributos.

V. CONCLUSIONES

VDM++ permite modelar de manera eficiente los sistemas reales que utilizan el paradigma de orientación a objetos que pueden tener impactos críticos para los usuarios y desarrolladores, tal como la propuesta de solución al caso de aplicación.

Es importante desarrollar un análisis sin ambigüedades para un sistema altamente riesgoso, también es necesario el desarrollo de las especificaciones formales y validarlas a través de una herramienta, tras la validación de las especificaciónes, la generación de código de programación permite incrementar la conformidad entre los requerimientos y el código que estará a cargo del equipo de desarrollo en la elaboración del sistema.

En este trabajo se presento las especificaciones formales del módulo de funcionalidades más relevantes en un sistema de control de paso de vehículos pesados a través de puentes. El usar especificaciones formales nos permitió reducir riesgos y aportar seguridad en el sistema desde la fase de análisis de la elaboración del sistema. Las especificaciones formales se desarrollaron un el lenguaje de especificación formal, VDM++, en conjunto con la herramienta que soporta este lenguaje, VDM++ ToolBox.

REFERENCIAS

[1] Fu, Gongkang & Hag-Elsafi, Osman. (2000). Vehicular Overloads: Load Model, Bridge Safety, and Permit Checking. Journal of Bridge Engineering-J BRIDGE ENG .5.10.1061/ (ASCE) 1084-0702 (2000) 5:1(49).

[2] Crespo-Minguillón, C. and Casas, J. A comprehensive traffic load model for bridge safety checking, In Structural Safety, Volume 19, Issue 4, 1997, Pages 339-359, ISSN 0167-4730

[3] Yochia Chen, Distribution of vehicular loads on bridge girders by the FEA using ADINA: modeling, simulation, and comparison, In Computers & Structures, Volume 72, Issues 1–3, 1999, Pages 127-139, ISSN 0045-7949, https://doi.org/10.1016/S0045-7949(99)00032-2.

[4] http://www.adina.com/

[5] Queille J.P. and Sifakis, J. "Specification and Verification of Concurrent Systems in Cesar," Proc. Fifth Intl. Symp. Programming, pp. 195-220, Lecture Notes In Computer Science 137. Springer-Verlag, 1981.

[6] http://www.cesar-lcpc.com/cesar-products.php

[7] Rushby, J."Formal methods and their role in the certification of critical systems", Technical Report CSL-95-1, SRI International, 1995.

[8] CSK SYSTEMS d. VDM++ Toolbox User Manual. Technical Report, 2009.

[9] Durr, E. and Katwijk, y J. V. "A Formal Specification Language For Object-
Oriented Designs", Proceedings 6th Annual European Computer Conference, Compeuro, 1992.

[10] Fitzgerald J. and Gorm Larsen, P. "Modelling Systems Practical Tools and Techniques in Software Development", 2a Ed, Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998.

[11] Fitzgerald, J., Gorm Larsen, P. y Mukherjee, P. and Verhoef, M. "Validated Designs for Object oriented Systems". Springer, New York, 2005

[12] Grupo de Trabajo de Armonización Global -Sistemas de Gestión de la Calidad - Proceso Guía de validación(GHTF / SG-3 / N99-10: 2004 (Edición 2) página 3

[13] Boehm, B. . "Software Risk Management". En Ghezzi, C .; McDermid, JA Actas de la 2ª Conferencia Europea de Ingeniería de Software. 1989.

[14] Mascia, N. y Sartorti. A. "Identificación y análisis de patologías en puentes de carreteras urbanas y rurales", Revista Ingeniería de Construcción Vol. 26 N°1, Abril de 2011 www.ing.puc.cl/ric PAG. 05-24

[15] Larsen, P. G., Battle, N., Ferreira, M., Fitzgerald, J., Lausdahl, K. and Verhoef. M. The Overture Initiative – Integrating Tools for VDM. ACM Software Engineering Notes, 35(1), January 2010.

[16] CSK SYSTEMS, “The VDM++ Language Manual”. Technical Report, 2009. v.1.0
[17] Jones, C. B. Systematic Software Development Using VDM, 2nd ed., Prentice Hall, 1990.
[18] Fitzgerald, J., Larsen, P. G., and Sahara, S. 2008. VDMTools: Advances in support for formal modeling in VDM. Sigplan Notices 43, 2 (Feb.), 3–11.