Verificación para el Desarrollo de un Sistema de Control de Semáforos Inteligente utilizando UPPAAL

[1]Luisa María Salhua Tincoso, Estudiante Universitario, [1]Freddy Miguel Valdivia Berrio, Estudiante Universitario, [1]Isabel Paredes Apaza, Estudiante Universitario, [1]Elizabeth Vidal Duarte, Mentora
[1]Universidad Nacional de San Agustín, Perú, lsalhuat@unsa.edu.com, iparedesap@unsa.edu.com fvaldiviab@unsa.edu.com

***Abstract*– La ciudad de Arequipa tiene el problema del congestionamiento en sus vías debido a que el sistema de semáforos que actualmente tiene la ciudad no ayuda a un tránsito fluido. Esto conlleva que las vías se congestionen y los habitantes pierdan valioso tiempo, creando malestar y disgusto. Al ver esta necesidad se propone el Desarrollo de un Sistema de Control de Semáforos. Dado que un Sistema de Control de Semáforos es considerado de carácter crítico es que se ha decidido utilizar técnicas matemáticas de verificación como Model Checking. La herramienta utilizada tanto para la simulación y verificación será UPPAAL. El principal aporte de nuestro artículo es mostrar el uso de técnicas matemáticas para el modelamiento de sistemas.**

***Keywords*-- Verificación, UPPAAL, semáforo, Sistema.**

I. Introducción

La tecnología cada día va avanzando mejorando la calidad de vida de las personas; la aplicación de nuevos sistemas inteligentes se hace más frecuente, transformando unos objetos simples a objetos relativamente más complejos y más interesantes. Sin embargo, existen sistemas que son importantes, y que un error en estos puede causar pérdidas de vidas. Para verificar este tipo de sistemas, existen técnicas basadas en matemáticas como Model Checking.

En el presente artículo realizaremos la propuesta de un nuevo sistema para el control de semáforos, para evitar las inmensas conglomeraciones que ocurren en las vías automovilísticas de la ciudad de Arequipa, acortando el tiempo malgastado al estar varado en estas congestiones, este sistema es verificado a través de Model Checking utilizando la herramienta UPPAAL.

El resto del artículo está organizado de la siguiente manera. En la sección II se hace referencia a los diferentes trabajos que tratan los problemas de Semáforos y sus soluciones. En la sección III realizamos una definición de los distintos conceptos que abarcamos para la realización del modelo de semáforos presentando y la investigación sobre la herramienta UPPAAL. En la sección IV definimos la propuesta de solución, requerimientos encontrados y el modelado en la herramienta utilizada. Finalmente presentamos nuestras conclusiones

Digital Object Identifier: (to be inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).

II. Trabajos Relacionados

La importancia de verificar propiedades en un modelo de un sistema, es un aspecto crucial en los sistemas críticos; mediante Model Checking, una forma de la verificación, se encuentra información acerca de modelos relacionados a semáforos. El artículo de Fitzgerald [1], presenta la especificación formal y un circuito implementado en la herramienta de verificación STOOLS para el diseño del controlador de semáforo asíncrono. Neudecker [7], trata de un semáforo virtual, utiliza un protocolo, que fue verificado mediante model checking. Finalmente, Bin Yu
[8] presenta un Traffic Light Control System (TLCS), sistema que es verificado por model checking acotado (BMC), mediante el uso de la herramienta BMC4PPTL.

A diferencia de los trabajos previamente desarrollados, nuestra propuesta trata de resolver una realidad local con características muy particulares, centrándonos en el cambio inteligente en los tiempos de los colores del semáforo, utilizando la herramienta de Model Checking UPPAAL para el modelamiento del sistema.

III. Verificación

A. *Definición*

La verificación nos da la evidencia que el diseño de una parte en particular del ciclo de vida del desarrollo del software cumple con los requisitos especificados, con los resultados obtenidos y así probar la coherencia, corrección e integridad del software [5].

B. *Model Checking*

Es un método para verificar formalmente sistemas concurrentes de estados finitos. Especificaciones sobre el sistema se expresan como fórmulas lógicas temporales, y los algoritmos eficientes simbólicas se utilizan para atravesar el modelo definido por el sistema y compruebe si la especificación mantiene o no [10].

Los pasos para realizar el Model Checking son:

1. Construir un modelo para el sistema: el conjunto de autómatas.
2. Formalizar las propiedades que se verificarán el uso de expresiones en la lógica.
3. Utilizar el corrector modelo (herramienta) para generar el espacio de todos los estados posibles y comprobar exhaustivamente si las propiedades se cumplen en todos y cada uno de los posibles comportamientos dinámicos del modelo.

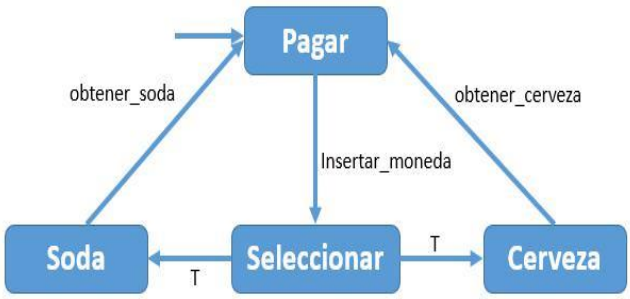

Fig. 1 Ejemplo Model Checking [10]

A modo de ejemplo (Fig. 1), se muestra las características que se toman en cuanto en Model Checking, para una máquina expendedora de gaseosa y cerveza. Representada por 4 estados (pagar, soda, seleccionar y cerveza), el estado inicial (pagar), indica el inicio del grafo presentado, que, dependiendo de las acciones y transiciones, como "insertar moneda" se puede pasar de un estado a otro, para finalmente terminar en un estado final; en este caso regresando al estado inicial.

C. Herramienta UPPAAL

UPPAAL es una herramienta con un ambiente integrado para modelar, la validación y la verificación de los sistemas en tiempo real modelados como redes de autómatas sincronizados, extendidas con los tipos de datos (números enteros, órdenes, etc. limitados) [6].

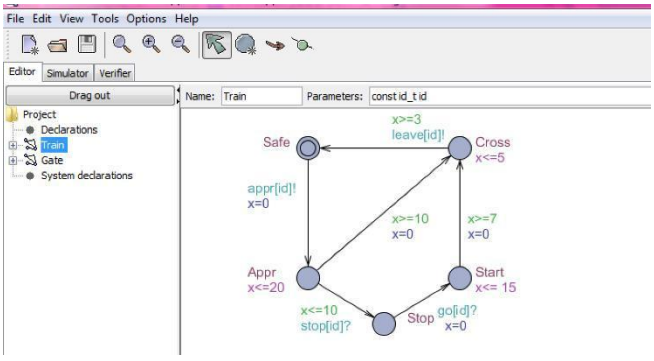

Fig. 2 Ejemplo de UPPAAL, sistema modelado con 5 estados

En (Fig. 2) podemos apreciar el entorno de UPPAAL, en el cual se encuentra la barra de herramientas donde tenemos opciones de crear un nuevo sistema, seleccionar los diferentes componentes del modelo, crear un nuevo estado, una nueva arista.

Existen tres modificadores para los estados de esquemas:

– Inicial (initial): El modificador initial determina cual es el estado donde comienza la ejecución. Cada esquema debe tener exactamente un estado inicial.

– Urgente (urgent): Si el estado está declarado como urgent, detiene el paso del tiempo mientras un proceso se encuentre en uno de ellos.

– Comprometido (committed): Los estados comprometidos al igual que los urgentes detienen el paso del tiempo, y además obligan a que la próxima transición elegida a ser ejecutada debe ser alguna transición habilitada saliente de (con origen en) un estado comprometido (establece una prioridad en la elección de las transiciones).

Los estados están conectados por medio de aristas que establecen la relación de transición entre los mismos. Las aristas pueden tener asociados: selectores, guardas, sincronizaciones y modificaciones.

– Selectores: Los selectores retornan, de manera no determinista, un valor en un rango especificado.

– Guardas: Las guardas establecen las condiciones necesarias para que una arista está habilitada. La especificación de una guarda se realiza por medio de expresiones lógicas con las mismas restricciones que las especificaciones de los invariantes.

– Sincronizaciones: Los procesos pueden sincronizarse a través de canales compartidos. Las acciones permitidas sobre los canales son: escribir (!) y leer (?). Cada arista solo puede hacer referencia a un único canal.

– Modificaciones: Cuando las aristas son ejecutadas, las expresiones establecidas en esta sección son evaluadas y su efecto cambia el estado del sistema. El lenguaje para especificar las transformaciones de estados es muy rico ya que, entre otros aspectos, permite invocar a funciones la cuales puede estar "programadas" con un estilo imperativo.

Nos ofrece también 3 pestañas dentro de la herramienta la primera llamada "Editor" nos dejará editar el modelado, la segunda llamada "Simulator" nos permite ver la transición de estados, y la última llamada "Vérifier" nos permitirá verificar si cumple con los requisitos especificados.

IV. APLICACIÓN: SEMÁFOROS

A. *Problema*

La ciudad de Arequipa es la segunda ciudad más importante del Perú, con 1, 15 millones de habitantes [11] y con parque automotor de 258 396 automóviles en circulación hasta el 2016 [12]. En Arequipa existen vías extensas que en horas puntas se congestionan frecuentemente, donde se encuentran un conjunto de semáforos a lo largo de las vías, los cuales tienen un tiempo establecido para cada estado del semáforo, creando un tiempo desperdiciado o muerto en algunas partes de la vía, que podría ser utilizado eficientemente (fig. 3 y 4).

Fig. 3 Aglomeración de autos, Av. Goyeneche, Arequipa

Fig. 4 Transito, Av. Ejército, Arequipa

Las figuras 3 y 4 muestran los problemas viales que existen en diferentes puntos cercanos al centro histórico de la ciudad.

B. *Solución propuesta*

En el artículo, realizamos la propuesta de un sistema inteligente de control de semáforos. Esta solución consiste en que cada uno de los semáforos dispongan de un sensor que enviará información al sistema si existe conglomeración de autos en su sección de control de la vía. Cuando el estado del semáforo sea verde; de haber congestión, se buscará secciones muertas en la vía, mediante el estado del sensor de los semáforos adyacentes próximos, el sistema autorregulará el tiempo de cada semáforo para el libre paso de los vehículos, cambiando a luz verde, si los sensores en alguna
parte del conjunto de semáforos, no detecta congestión en estado verde, el sistema tornara a los tiempos establecidos originales.

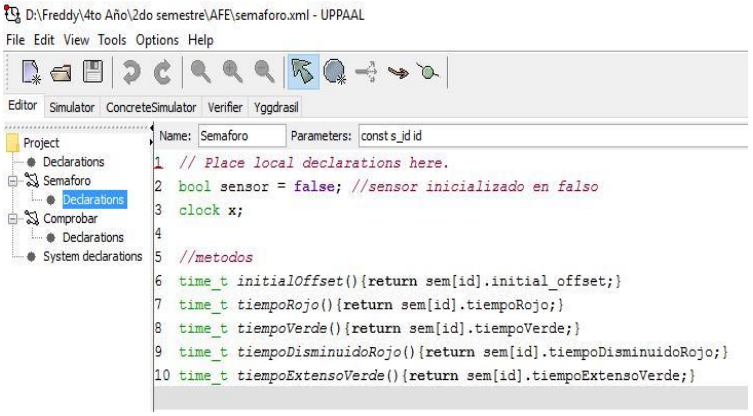

Fig. 5 Declaraciones

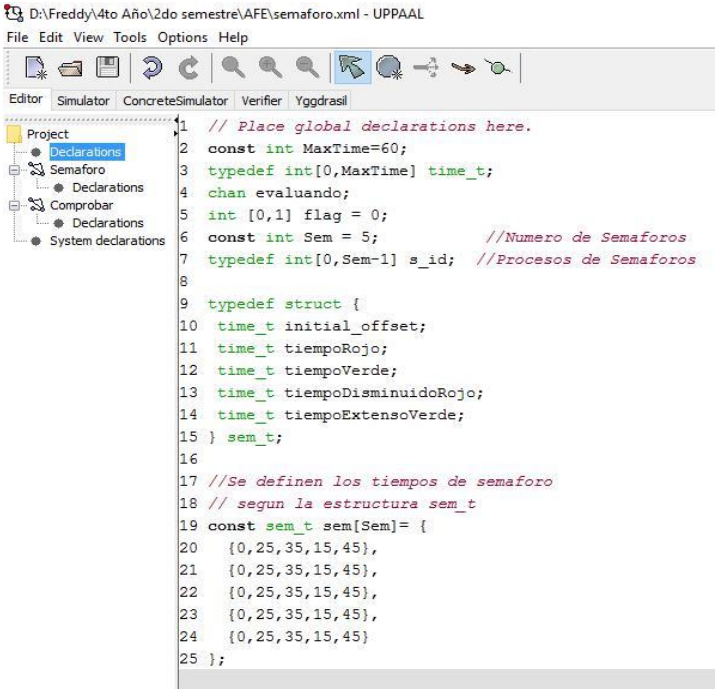

Fig. 6 Semáforo

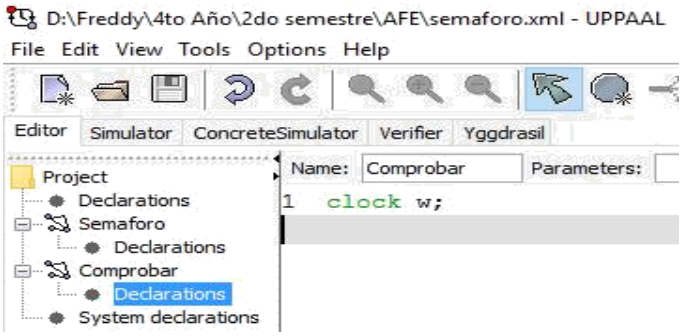

Fig. 7 Comprobar

C. Requerimientos

Los requerimientos identificados se detallan a continuación:

R1. Un sistema desarrollado en computadora para gestionar el cambio de tiempo de los semáforos.

R2. El sensor tiene dos estados: true y false

R3. El estado true en el sensor significa congestionamiento

R4. El estado false en el sensor significa libre de congestión

R5. El semáforo tendrá 2 modos de tiempo: normal y extenso

R6. El semáforo sólo considerara 2 colores: verde y rojo

R7. Asignación de sensores sea igual al número de semáforos

R8. Cambiar el estado a modo extenso del semáforo en caso de congestionamiento cuando el estado del sensor sea true;

R9. Cambiar el estado a modo normal del semáforo en caso de congestionamiento cuando el estado del sensor sea false;

R10. Mostrar los cambios efectuados en los semáforos.

D. Modelo en UPPAAL

Se detallará a continuación el modelo realizado en la herramienta.

En (Fig. 8 y 9), se muestra cómo quedaría el sistema en la herramienta UPPAAL, cumpliendo el requerimiento R1.

En Semáforo (Fig. 10), se inicializa el sensor con false (Línea 2). En esa línea de código se cumple el requerimiento R4, también haciendo alusión que el sensor tendrá 2 estados; true y false; cumpliendo también el requerimiento R2 en la

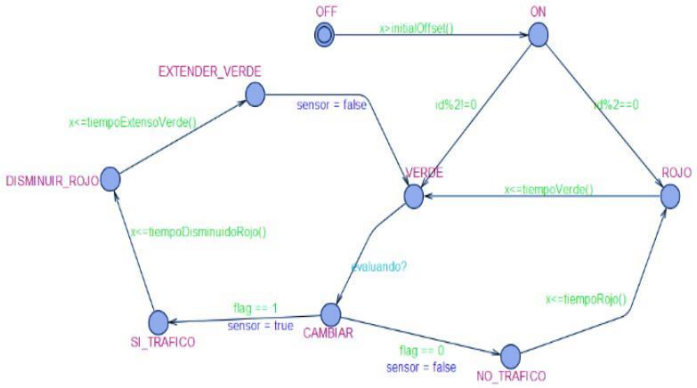

Fig. 8 Sistema de Control de Semáforo

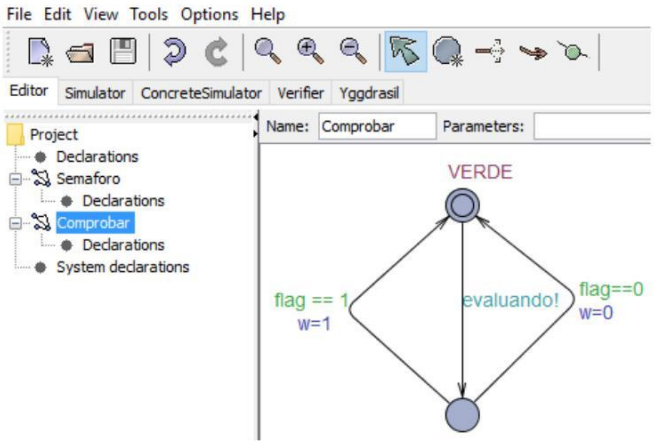

Fig. 9 Comprobar

```
1  // Place local declarations here.
2  bool sensor = false; //sensor inicializado en falso
3  clock x;
```

Fig. 10. Requerimiento2 y 3

El estado se actualizará a true si existe congestión y estará en false en caso contrario (Fig. 8).

En Semáforo también se declara sus métodos, donde tendrá el método de tiempo rojo y tiempo verde, donde está el tiempo de duración normal del semáforo, y también tiene el método tiempo extenso del color verde y el tiempo disminuido rojo, que compensara el aumento de tiempo de color verde, cumpliendo el requerimiento R5; también sé que se considera los 2 colores verde y rojo; cumpliendo el requerimiento R6.

```
5  //metodos
6  time_t initialOffset(){return sem[id].initial_offset;}
7  time_t tiempoRojo(){return sem[id].tiempoRojo;}
8  time_t tiempoVerde(){return sem[id].tiempoVerde;}
9  time_t tiempoDisminuidoRojo(){return sem[id].tiempoDisminuidoRojo;}
10 time_t tiempoExtensoVerde(){return sem[id].tiempoExtensoVerde;}
```

Fig. 11 Requerimiento 5 y 6.

En declaraciones (Fig. 12), se mostrará que se tendrá un número de semáforos, junto con Semáforo. En declaraciones (Fig. 10), se detalla que cada semáforo tendrá su propio sensor; ya mostrados en los anteriores cumplimentos de requerimientos; realizando también el requerimiento R7.

En declaraciones

```
5  int [0,1] flag = 0;
6  const int Sem = 5;            //Numero de Semaforos
7  typedef int[0,Sem-1] s_id;    //Procesos de Semaforos
8
```

Fig. 12 Requerimiento 7

En Semáforo (Fig. 13) y el modelo en UPPAAL (Fig. 8 y 9), se colocó que, cuando exista congestionamiento se cambie a su estado extenso, pero si no existe el congestionamiento, estará en su estado normal, y mostrándolo en el semáforo, cumpliendo los requerimientos R8, R9 y R10.

```
2  bool sensor = false; //sensor inicializado en falso
3  clock x;
4
5  //metodos
6  time_t initialOffset(){return sem[id].initial_offset;}
7  time_t tiempoRojo(){return sem[id].tiempoRojo;}
8  time_t tiempoVerde(){return sem[id].tiempoVerde;}
9  time_t tiempoDisminuidoRojo(){return sem[id].tiempoDisminuidoRojo;}
10 time_t tiempoExtensoVerde(){return sem[id].tiempoExtensoVerde;}
```

Fig. 13 Código de Semáforo

En declaraciones (Fig. 14) colocamos cuanto es el tiempo máximo (líneas 2 y 3). Se presentan cuántos son los semáforos existentes para el ejemplo y los procesos creados para estos mismos (línea 6 y 7). También definimos como será la estructura del semáforo (líneas del 9 al 15) y por último se define los tiempos (líneas 19 al 24).

```
1  // Place global declarations here.
2  const int MaxTime=60;
3  typedef int[0,MaxTime] time_t;
4  chan evaluando;
5  int [0,1] flag = 0;
6  const int Sem = 5;            //Numero de Semaforos
7  typedef int[0,Sem-1] s_id;    //Procesos de Semaforos
8
9  typedef struct {
10  time_t initial_offset;
11  time_t tiempoRojo;
12  time_t tiempoVerde;
13  time_t tiempoDisminuidoRojo;
14  time_t tiempoExtensoVerde;
15 } sem_t;
16
17 //Se definen los tiempos de semaforo
18 // segun la estructura sem_t
19 const sem_t sem[Sem]= {
20    {0,25,35,15,45},
21    {0,25,35,15,45},
22    {0,25,35,15,45},
23    {0,25,35,15,45},
24    {0,25,35,15,45}
25 };
```

Fig. 14 Código de Declaraciones

E. Validación

UPPAAL tiene una funcionalidad llamada Simulación, donde se puede ver cómo trabaja el sistema y seguir por los estados que pasa y cómo se produce los cambios especificados.

A continuación, se muestra una parte del modelo de cómo es la simulación del sistema

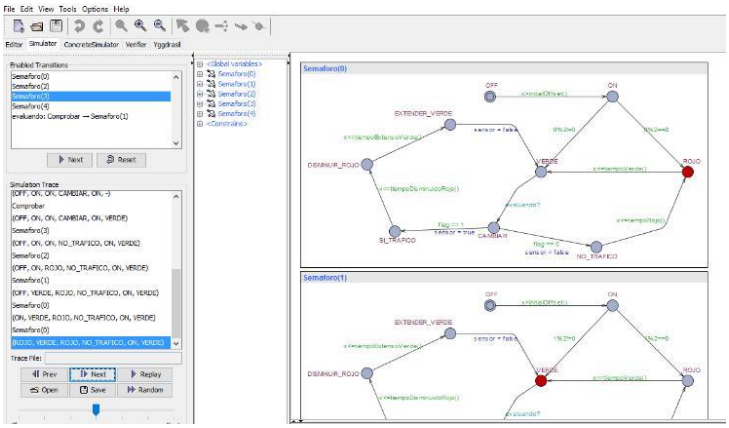

Fig. 15 Simulación de Semáforo 0 y 1

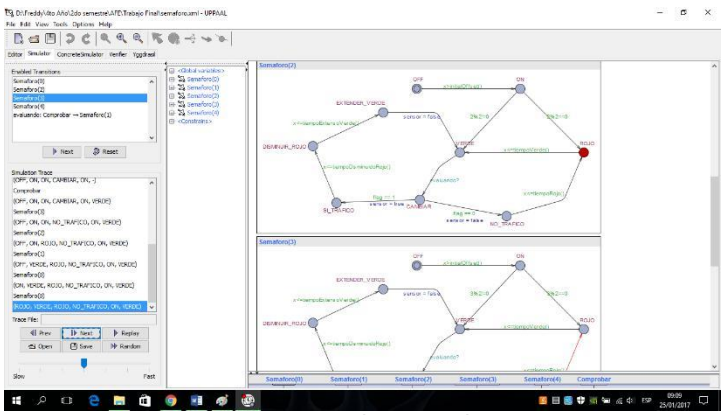

Fig. 16 Simulación de Semáforo 2 y 3

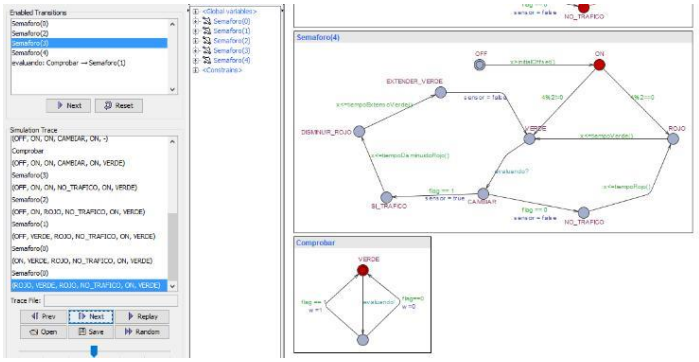

Fig. 17 Simulación de Semáforo 4 y de Comprobar

Como se muestra (Fig. 15, 16 y 17), se crean los 5 semáforos que declaramos anteriormente en las líneas de código (Fig. 14), junto con Comprobar que estará verificando si existe el congestionamiento o no, los semáforos funcionan sincronizadamente, si detectan algún cambio en su sensor, podrán cambiar a modo extenso o si no existe ningún cambio en el sensor, continuaran en su modo normal, como se muestra (Fig. 18 y 19).

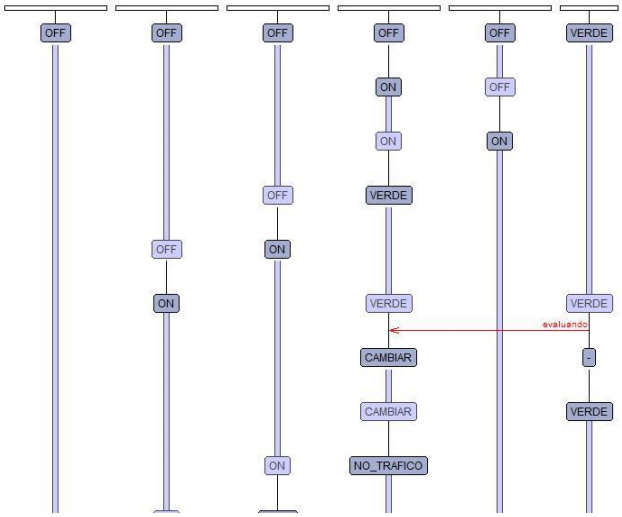

Fig. 18. Simulación de Semáforos y Comprobar

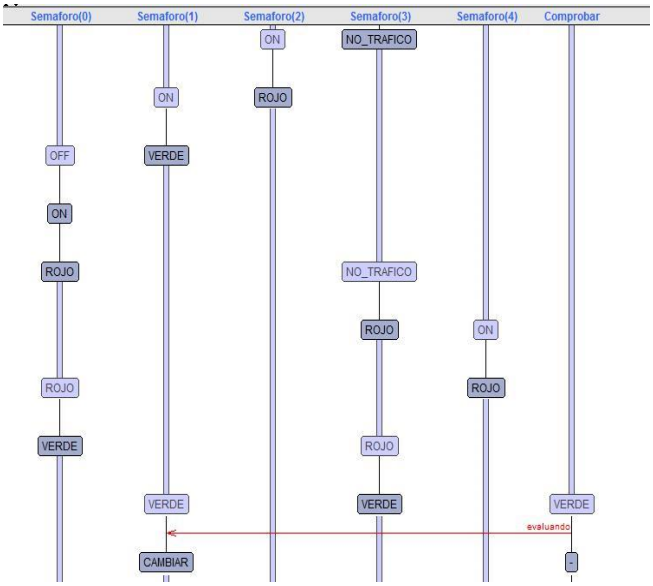

Fig. 19 Simulación de Semáforos y Comprobar

Cada vez que un semáforo pida evaluar la existencia de congestionamiento, irán a Comprobar, esto sucederá cada vez que lo semáforos pasen por la petición evaluando.

V. Discusión

El sistema de control de tráfico, es una solución al problema cotidiano de nuestra ciudad, que podría ser aplicado a cualquier otra ciudad del mundo, nuestro trabajo es una muestra simple que aborda desde otro punto de vista, a comparación de los trabajos relacionados, brindando una solución a los problemas de tráfico, en las vías extensas.

Además, aplicando Model Checking en nuestro modelo, sirvió para poder hacer las verificaciones pertinentes al sistema, junto con la herramienta UPPAAL, con su sistema de simulación, gráficas, características realmente impresionantes y en tiempo real, sirvió para el mejoramiento de nuestro Sistema de Control de Semáforos.

Referencias

[1] J. Fitzgerald, P. Gorm Larsen, P. Mukherjee, N. Plat, M. Verhoef, "Validated Designs for Object-oriented Systems" 2004, pp.142-163 .

[2] J. Malla Reddy, S.V.A.V Prasad, "The role of Verification and Validation on Software Testing" , International Conference on Computing for Sustainable Global Development,INDIA, 2016 978-3805-4421-2/16

[3] P. Arcaini, S. Bonfanti, A. Gargantini, A. Mashkoor and E. Riccobene, "Formal Validation and Verification of a Medical Software Critical Component" 978-1-5090-0237-5/15, 2015.

[4] J. Fitzgerald and P. Gorm Larsen, *"Modelling Systems: Practical Tools and Techniques in Software Development"*, Second Edition.. Cambridge University Press, Cambridge u. a. 1998, ISBN 0-521-62605-6.

[5] ISO/IEC Guide 99:2007, "International vocabulary of metrology-Basic and general concepts and associated terms(VIM)", 2008

[6] Lau K. and W. Z., "Software component models", IEEE Trans. Software Eng., 33(10), 709–724, 2007.

[7] T. Neudecker, N. An, Hannes Hartenstein, "Verification and evaluation of fail-safe Virtual Traffic Light applications" 2014

[8] Bin Yu, Z. Duan, C. Tian,"Bounded Model Checking of Traffic Light Control System", 2014

[9] C. Mellon, "Model Checking" [online], https://www.cs.cmu.edu/~modelcheck/

[10] C. Barrier, J. Peter, "Principles of Model Checking"

[11] Departamento de Arequipa, Wikipedia, https://es.wikipedia.org/wiki/Departamento_de_Arequipa

[12] 258 mil vehículos circulan en Arequipa, Diario El Pueblo, http://elpueblo.com.pe/noticia/primera/258-mil-vehiculos-circulan-en-arequipa