Modelado, Prueba y Verificación De Sistemas Distribuidos Usando RTDS e IFx

Juan Pablo Girón Ruiz
Pontificia Universidad Javeriana Cali
jpgironruiz@gmail.com

Resumen

One of the contemporary biggest challenges when designing Hardware/Software systems is guaranteeing that they are free of defects. The use of formal techniques like Model Checking allows ensuring that systems satisfy a given property because of exhaustive verification; however, they do not guarantee that the system has no errors. On the other hand, the use of functional black box testing, provides a notion of how well designed the system is, but this approach is not the best way to evaluate the critical modules of a distributed system.

This paper is subdivided into three major parts in order to explain a methodology for minimizing errors that combines two techniques, one is formal and the other one is semiformal on a case study corresponding to the parking system at the Pontificia Universidad Javeriana Cali, PUJC. Using PragmaDev's Real Time Developer Studio (RTDS), the model was described using Message Sequence Charts (MSC) and the Specification and Description Language (SDL); its simulation tool and the Testing and Test Control Notation version 3 (TTCN-3) were used for black box functional testing. Finally, by using the Verimag's IF language and the IFx toolset, Model Checking techniques were applied to formally verify properties expressed by observers on a critical component of the system.

palabras claves: Modelling of distributed systems, functional testing, Model Checking, formal verification, SDL, MSC, TTCN-3, IF, RTDS, IFx toolset.

I. Introducción

LA COMPLEJIDAD de los sistemas tanto en Hardware como en Software se ha venido incrementando significativamente y por ende la probabilidad que los sistemas presenten fallas aumenta [1]. Diversas áreas de las ciencias de la computación y la ingeniería han propuesto diferentes técnicas que permiten obtener sistemas Hardware/Software con un mínimo de errores; dichas técnicas estructuran la descripción del sistema a través de métodos formales o no formales, los cuales se pueden evaluar implementando desde una prueba de caja negra hasta el uso de métodos formales.

A pesar que existen métodos de verificación, validación y prueba de modelos, la complejidad de los sistemas se ha venido incrementando con gran rapidez y la forma de detectar errores en sistemas críticos con métodos separados es insuficiente [2].

Este trabajo pretende valorar el proceso de verificación del correcto funcionamiento de un modelo que representa un sistema distribuido, empleando un caso de estudio a partir de la unión de dos técnicas: métodos formales y pruebas funcionales de caja negra, en diferentes etapas del desarrollo de sistemas usando las herramientas RTDS e IFx. Para lograr el anterior objetivo, se estructuró la investigación con el diseño del modelo del caso de estudio usando el lenguaje SDL, el cual fue verificado a través de pruebas funcionales de caja negra usando el lenguaje TTCN-3. Finalmente, se verificó formalmente algunas propiedades del modelo del caso de estudio, por medio de la técnica Model Checking sobre la herramienta IFx.

II. Fundamentación Teórica

I. Modelado

El uso de un modelo o especificación formal permite disminuir los errores dentro del desarrollo de los sistemas [3]. Especificar un sistema por medio de lenguajes formales tiene muchas ventajas entre las cuales encontramos: Obtener modelos con especificación y descripción clara, automatizar la fase de verificación del diseño por medio de pruebas o verificaciones formales.

El lenguaje de especificación y descripción, SDL, fue usado para modelar el sistema de parqueadero de la Universidad Javeriana Cali. Dado a que, permite realizar pruebas funcionales de caja negra y posteriormente verificaciones formales usando la herramienta IFx.

En el desarrollo de sistemas Hardware/Software encontramos diferentes etapas en las cuales es necesario plantear un requerimiento de especificación clara y concisa, con el fin de elaborar el diseño del mismo. Message Sequence Charts (MSC)[1] es un lenguaje textual y gráfico que permite expresar requerimientos de especificaciones, simulación y validación, además de describir el comportamiento de comunicación entre las entidades del sistema y el ambiente [4]. Actualmente existe una relación directa entre la especificación realizada en un diagrama MSC y SDL, que facilita la parte de diseño del modelo.

II. Software de Pruebas

Una prueba es un conjunto de actividades que tiene como objetivo identificar fallas en un sistema y evaluar su nivel de calidad, para obtener la satisfacción del usuario. Esto es un conjunto de tareas con metas claramente definidas [5].

Mayoritariamente se emplean dos métodos de pruebas que son: Caja Negra (En inglés: Black-Box) y Caja blanca (En inglés: White-Box). El primer método está basado en los requerimientos y especificaciones para saber si el sistema es correcto, pero no es necesario conocer el mismo en detalle. El segundo hace parte de la categoría basada en la estructura: el caso de prueba se deriva desde el código fuente interno del modelo; la hipótesis fundamental consiste en que el modelo cumple con los requerimientos del cliente [5, 6].

Testing and Test Control Notation Version 3, más conocido por sus siglas en inglés como TTCN-3 es un lenguaje de especificación e implementación de pruebas de tipo caja negra para sistemas distribuidos y reactivos [7]. TTCN-3 posibilita la interacción con otros lenguajes de descripción, por ejemplo SDL [7, 8].

Una de las características mas relevantes del lenguaje TTCN-3 es que tiene integrado un mecanismo de coincidencia que permite evaluar si el sistema satisface ciertas condiciones, definidas en los casos de prueba. En TTCN-3 se encuentra los siguientes valores para el veredicto de una prueba: **pass**, que significa que el SUT se comportó de acuerdo al propósito de la prueba,**inconc**, que significa que no se puede determinar si el SUT pasó o falló la prueba, **fail**, que indica que el SUT no cumplió con el propósito de la prueba, **error**, esta asignación la hace el ambiente de ejecución de TTCN-3 cuando hay fallas en el componente de pruebas o en el SUT y finalmente **none**, que es el valor inicial, cuando el veredicto no ha sido asignado.

III. Verificación Formal

Los métodos formales según [9] son lenguajes, técnicas y herramientas basados en estructuras matemáticas con el objetivo de especificar y verificar sistemas. El uso de formalismos en la

[1] Información consultada el 20 de Mayo del 2014 en el siguiente sitio: `http://www.sdl-forum.org/MSC/index.htm`.

verificación de sistemas no garantiza que estos estén libres de errores, pero brinda la posibilidad de expresar modelos con un mejor entendimiento.

La verificación de hardware es la demostración que un circuito o un sistema (Nivel de implementación) se comporta de acuerdo a un conjunto de requerimientos (Nivel de especificación) [10]. La verificación formal es contraria a la simulación, en el sentido que no es necesario crear un conjunto de estímulos al sistema para garantizar su comportamiento. La simulación es un método poco práctico, debido a que es practicamente imposible lograr estimular el circuito o el sistema con todas las posibles entradas que va a tener durante su funcionamiento.

Model Checking es una técnica automática [11] de verificación formal que depende de la construcción de un modelo finito del sistema y verifica si una propiedad deseada se satisface en dicho modelo [9]. La exploración de todos los posibles estados del sistema se da de forma exhaustiva. Esta técnica permite mostrar a través de contraejemplos los errores del modelo, describiendo el camino desde el estado inicial del sistema al estado que viola la propiedad que está siendo verificada, lo que facilita la depuración de fallas en el diseño del sistema [12, 11].

IV. Herramienta IFx

La herramienta IFx fue desarrollada por investigadores de Verimag en Francia. Esta herramienta es un ambiente para el modelado, validación y verificación de sistemas [13].

La herramienta IFx posee características que proveen grandes ventajas a los diseñadores de sistemas tales como; soportar lenguajes de modelado de alto nivel como SDL, UML, adicionalmente existe una traducción directa con dichos lenguajes de modelado. Los modelos expresados en el lenguaje IF pueden ser optimizados con herramientas de análisis estático que posee la herramienta IFx. Finalmente las propiedades que se desean verificar se pueden expresar por medio de observadores.

Los tipos de observadores que fueron usados en este trabajo es de tipo **cut**, dado a que permite verificar sólo una parte del sistema que involucra la propiedad de interés. Lamentablemente, existe poca literatura sobre el uso de observadores de tipo intrusivo, a lo largo del desarrollo de las verificaciones formales al caso de estudio se trató de pedir mayor información al equipo Verimag sobre la utilización de dichos observadores, pero nunca hubo respuesta por parte de ellos.

III. Resultados

I. Modelado

El sistema de parqueo de la Pontificia Universidad Javeriana, se modeló satisfactoriamente usando los lenguajes MSC y SDL, para la especificación y descripción del sistema respectivamente. La figura 1 representa cómo el sistema de parqueo de la PUJ-Cali está compuesto.

Basicamente, el sistema de parqueo de vehículos cuenta con C controladores de zonas que a su vez pueden tener hasta Z zonas; éstas a su vez possen un sistema de sensores (S) capaces de informar a su controlador si un vehículo ingresó o salió de dicha zona. Adicionalmente, el sistema puede tener hasta E entrada y S salidas principales, con sus respectivos sensores tanto para el ingreso (SE) o salida de vehículos (SS).

El sistema de parqueadero cuenta con una base de datos, como se aprecia en la figura 1(a), dónde se encuentra almacenada la información de los usuarios que pueden ingresar o salir del sistema de parqueo. Dado que no es de interés verificar el correcto funcionamento de la base de datos, ésta se modeló como una simple lista que dado un valor, que corresponde a un código de usuario, retorna si está habilitado para el ingreso o salida del sistema de parqueo.

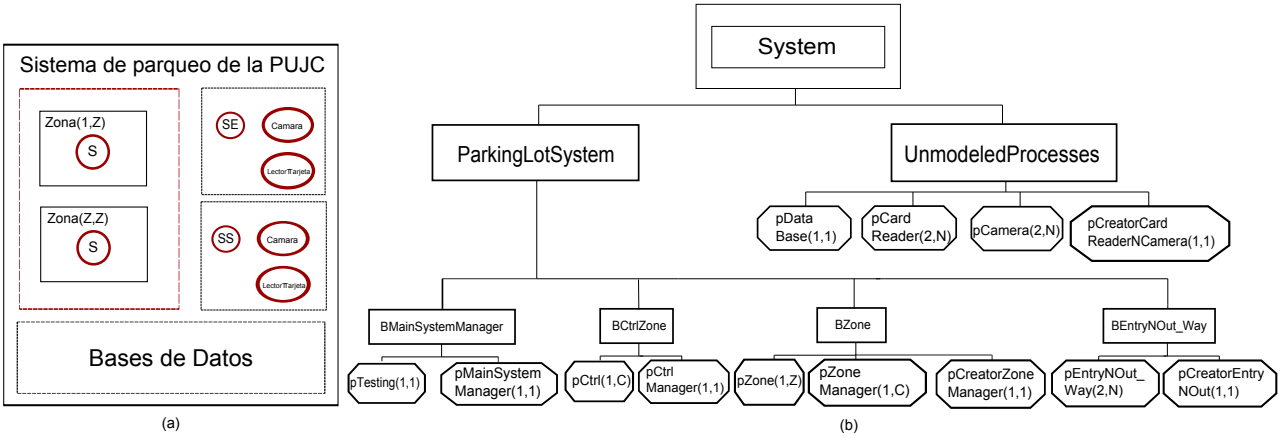

Figura 1: *(a) Componentes del Sistema de Parqueo de la PUJC (b) Arquitectura Gral. del sistema de parqueo de la PUJC.*

La metodología para el diseño de las especificaciones fue Top-Down, donde se parte de un alto nivel de abstracción del sistema a un nivel mas fino. Por otra parte el diseño del sistema se hizo usando la metodología Bottom-Up, donde se parte diseñando módulos sencillos que a su vez la unión de éstos forman bloques con un nivel de abstracción mas alto.

El sistema diseñado es dinámico lo que conlleva que es posible ampliar el sistema (Zonas, controladores de zonas, entradas y salidas principales) a través de una secuencia de mensajes que permiten que procesos específicos creen y enlacen diferentes procesos que permiten cumplir con las especificaciones del sistema, por ejemplo; El sistema puede tener mas entradas y salidas principales, a su vez éstas salidas y entradas siempre deberán de tener asociado un lector de carnés y una cámara para validar si un usuario puede ingresar o no al sistema, en el momento de la creación de dichos procesos se enlazan entre sí para formar un entrada o salida principal de manera correcta.

Como se aprecia en la Figura 1(b), el primer bloque, ParkingLotSystem, está conformado por los procesos y elementos que se modelaron con un mayor nivel de detalle; por el contrario, en el segundo bloque se encuentran los elementos del sistema que no se modelaron sino que se han considerado como procesos simples, los cuales reciben un valor y retornan otro pero sin realizar un procesamiento específico de los datos, dado que no fue de interés del proyecto modelarlos.

II. Pruebas

Las pruebas funcionales de caja negra fueron implementadas de tres maneras diferentes: La primera fue por medio de la herramienta RTDS que permite enviar estímulos al sistema bajo prueba, lo anterior se consigue hacer de manera muy rápida y sencilla lo que sirve para determinar de manera superficial sí el sistema posee problemas. La segunda forma en que se implementaron las pruebas funcionales, fue a través de interfaces gráficas que proporciona la herramienta RTDS, a diferencia del primer método usado, éste posee un mecanismo básico de coincidencia lo que permite establecer la señal para ejercitar el sistema y un valor esperado. A pesar, que las pruebas se pueden realizar de manera sencilla, RTDS no proporciona mecanismos de coincidencia sofisticados ni mucho menos de automatización, lo que se convierte en una herramienta poco deseable para determinar sí el sistema se encuentra bien construido. La figura 2, muestra los diferentes métodos de pruebas integrados en la herramienta RTDS.

Finalmente, la tercera forma que se usó para las pruebas funcionales fue a través del lenguaje

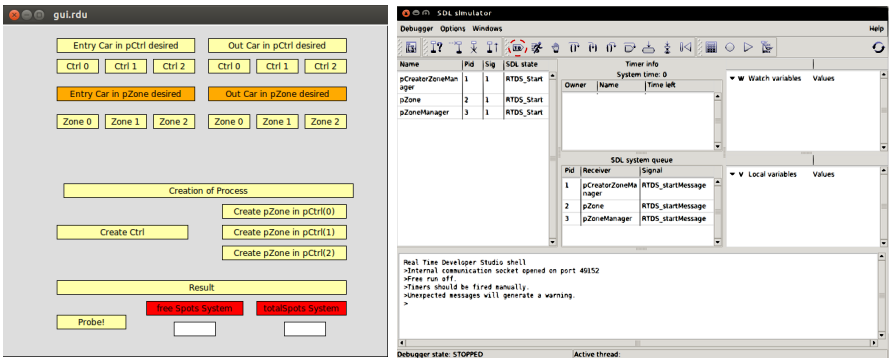

Figura 2: *Módulo intregrado para hacer pruebas en la herramienta RTDS. (a) Envío y recepción del modelo por medio de una interfaz gráfica (b) Envío de señales al modelo*

TTCN-3. La figura 3 muestra básicamente el objetivo de la prueba, en la parte (a) muestra las condiciones iniciales del sistema, donde posee cuatro zonas dos de ellas asociadas a un controlador de zonas y las restantes a otro. La parte (b) se describen los valores con los cuales se pretende ejercitar el sistema, donde se establece fundamentalmente la cantidad de carros que van a ingresar y los que van a salir en una hora determinada. Finalmente se puede estimar el valor esperado de plazas libres que debe de tener cada zona por horas. La parte (c) de la figura 3 muestra como se ejecutaron los casos de pruebas, lo que indica la secuencia de envío de mensajes, es que se pretende estimular al sistema ingresando y sacando vehículos, en este caso se hizo hasta la hora *6:50 a.m.*, que corresponde al indice *19*, una vez enviados los estímulos al sistema, se hace un requerimiento de información a la zona de interés, nuevamente para este caso, se trata de la zona 0 del controlador 0. Según la tabla mostrada en la figura 3(b) se espera que las plazas libres de dicha zona sean 156. El modelo satisfizo el objetivo de la prueba, lo que da una noción más profunda de que el sistema estaba bien construido.

III. Verificación Formal

Dado a que las pruebas funcionales de caja negra no son apropiadas para verificar propiedades que se consideren críticas en un sistema, se hace uso de la técnica Model Checking sobre la herramienta IFx. Aprovechando la herramienta RTDS que permite trasladar un modelo escrito en SDL al lenguaje IF, se hizo la conversión. Infelizmente, surgieron muchos errores en el momento de verificar las propiedades en cuestión; el problema básicamente surgía porque la herramienta RTDS no soportaba ciertos tipos de modelado para realizar la conversión. Inmediatamente se le informó al equipo *PragmaDev* dicho problema, la respuesta por parte de ellos fue que van a incluir nuestros comentarios en su nueva versión, mitigando ese tipo de problemas.

Como se aprecia en la figura 1, unos procesos que se podría considerar crítico es *pZone*, dado a que éste debe de reportar la cantidad exacta de zonas disponibles, de no hacerlo la información suministrada al usuario será errada, y por ende el sistema no funcionará. Para verificar la ausencia de *deadlocks* en el proceso *pZone*, se construyó un observador tipo *cut* que expresaba la siguiente propiedad: *"Independiente de cuál sea la señal de entrada al proceso pZone, éste ejecutará las transiciones correspondientes y evolucionará al estado Idle"*.

En la documentación de la herramienta IFx, por parte de Verimag, hablan que ésta posee un analizador estático que permite la detección de *deadlocks* en el modelo en IF, en el desarrollo de este proyecto fue imposible obtener una respuesta sobre la correcta utilización de dicha herramienta,

CONDICIONES DEL SISTEMA		
PLAZAS TOTALES DEL SISTEMA	Cantidad de Zonas	Plazas libres por Zona
1200	4	300

(a)

Hour	In Cars	Out Cars	Zona 0 Ctrl0	Zona 1 Ctrl0	Zona 0 Ctrl1	Zona 1 Ctrl1	Free Spots System	Index
5:15	0	0	300	300	300	300	1200	0
5:20	0	0	300	300	300	300	1200	1
5:25	0	0	300	300	300	300	1200	2
5:30	1	0	299	300	300	300	1199	3
5:35	0	0	299	300	300	300	1199	4
5:40	0	0	299	300	300	300	1199	5
5:45	0	0	299	300	300	300	1199	6
5:50	0	0	299	300	300	300	1199	7
5:55	0	0	299	300	300	300	1199	8
6:00	0	0	299	300	300	300	1199	9
6:05	0	0	299	300	300	300	1199	10
6:10	4	1	296	300	300	300	1196	11
6:15	2	1	295	300	300	300	1195	12
6:20	4	0	291	300	300	300	1191	13
6:25	15	2	278	300	300	300	1178	14
6:30	12	5	271	300	300	300	1171	15
6:35	30	2	243	300	300	300	1143	16
6:40	26	3	220	300	300	300	1120	17
6:45	36	2	186	300	300	300	1086	18
6:50	37	7	156	300	300	300	1056	19
6:55	30	2	128	300	300	300	1028	20
7:00	32	3	99	300	300	300	999	21
7:05	7	0	92	300	300	300	992	22
7:10	54	6	44	300	300	300	944	23
7:15	36	5	13	300	300	300	913	24
7:20	26	9	9	287	300	300	896	25
7:25	18	1	1	278	300	300	879	26

(b)

```
/*Loop for In and Out of Cars to the Parking System*/


indexHour:=19;

for (index:=0;index<4*indexHour+4;index:=index+1)
{
  /* In cars */

  numCars:=aEntryCar[index];
  nCtrl_Entry:=aCtrlEntryCar[index];
  nZone_Entry:=aZoneEntryCar[index];
  for (count_numCars:=0; count_numCars<numCars;count_numCars:=count_numCars+1)
  {
      execute(tc_EntryCar(nCtrl_Entry,nZone_Entry));
      t_waitEntryCar.start(8);
      t_waitEntryCar.timeout;
  }
  /* Out Cars */
  numCars:=aOutCar[index];
  nCtrl_Entry:=aCtrlOutCar[index];
  nZone_Entry:=aZoneOutCar[index];
  for (count_numCars:=0; count_numCars<numCars;count_numCars:=count_numCars+1)
  {
      execute(tc_OutCar(nCtrl_Entry,nZone_Entry));
      t_waitEntryCar.start(8);
      t_waitEntryCar.timeout;
  }


}
numCars:=aExpectedSpots[4*indexHour];
nCtrl_Entry:=aCtrlExpected[4*indexHour];
nZone_Entry:=aZoneExpected[4*indexHour];
execute(tc_VerifyFreeSpots(nCtrl_Entry,nZone_Entry,numCars));
```

(c)

Figura 3: *Diseño de la prueba usando TTCN-3. (a) Condiciones iniciales del sistema (b) Valores para ejercitar el sistema con sus valores esperados (c) ejecución de casos de pruebas para la verificación de plazas libres en una zona después del ingreso y salida de vehículos.*

dado a que lo que hacía dicha utilidad era eliminar todo el código y siempre sobre escribía el sistema a un par de estados con condiciones ideales, mas no representaba el sistema inicial. Por lo anterior, se diseñó una propiedad que básicamente evaluaba que por cada estímulo ingresado el proceso siempre debía regresar al estado Idle. De esa forma, se verificó la ausencia de deadlocks en el proceso pZone.

Otra propiedad que se deseaba verificar era: *"Una zona siempre reportará sí un vehículo ha ingresado enviando una señal llamada sEntered_Car; sí y sólo sí, se reciben las siguientes secuencias de sensores: sIR1_Zone, sIR2_Zone y sLoopInductive o sIR4_Zone, sIR3_Zone y sLoopInductive"*. Nuevamente el interés radica en el proceso pZone, la anterior propiedad no se satisfizo con el diseño inicial, debido a que, en el diseño del sistema en SDL en un mismo estado estaba habilitado para recibir señales de ingreso o salida de vehículos y lo que realmente se esperaba es que en cada estado se deseará una sola acción al tiempo. El anterior defecto por medio de pruebas funcionales no era posible detectarlo ya que se hubieran generado la secuencia de señales exactas para simular el ingreso o salida de un vehículo. Gracias a las técnicas formales como Model Checking fue posible entregar un sistema con mínimo de errores.

IV. Conclusiones

La utilización de lenguajes formales y semi-formales para la especificación y diseño de modelos, permiten especificar sistemas de manera más precisa que luego podrán ser verificados haciendo uso de pruebas funcionales definiendo la característica de la prueba a partir de las especificaciones definidas del sistema; además, se puede extender su verificación haciendo uso de técnicas formales como Model Checking para garantizar que el sistema cumpla con sus propiedades previamente especificadas.

Por otro lado, es importante destacar que las pruebas funcionales de caja negra y las verificaciones formales son complementarias; si bien es cierto que hacer uso de técnicas formales como es el caso de Model Checking proporciona un análisis exhaustivo al diseño, lo que permite garantizar que el sistema cumple con cierta propiedad; pero su aplicabilidad a sistemas que posean un alto nivel de abstracción se ve afectado significativamente por la complejidad del espacio de estados.

A pesar que la herramienta RTDS presenta algunas falencias en la construcción de pruebas funcionales sobre el lenguaje TTCN-3 y la traducción del lenguaje SDL a IF, no deja de ser una alternativa atractiva para la construcción de sistemas distribuidos abarcando parcialmente todo el ciclo de vida de la construcción de sistemas distribuidos.

Referencias

[1] K. Schneider, *Verification of Reactive Systems: Formal Methods and Algorithms*. Springer, 2004. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=Z92bL1VrD_sC&pgis=1

[2] J. Bowen, K. Bogdanov, J. Clark, M. Harman, R. Hierons, and P. Krause, "FORTEST: formal methods and testing," in *Proceedings 26th Annual International Computer Software and Applications*. IEEE Comput. Soc, 2002, pp. 91–101. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1044538

[3] R. M. Hierons, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, H. Zedan, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, and K. Kapoor, "Using formal specifications to support testing," *ACM Computing Surveys*, vol. 41, no. 2, pp. 1–76, Feb. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1459352.1459354

[4] M. Ebner, "TTCN-3 Test Case Generation from Message Sequence Charts," in *In Workshop on Integrated-reliability with Telecommunications and UML Languages (ISSRE04:WITUL)*, 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.109.6402

[5] B. Homès, *Fundamentals of Software Testing*. John Wiley & Sons, Inc, 2013. [Online]. Available: http://books.google.com.co/books/about/Fundamentals_of_Software_Testing.html?id=ZqoaFElPJLgC&pgis=1http://dx.doi.org/10.1002/9781118602270.ch5

[6] P. Ammann and J. Offutt, *Introduction to Software Testing*, 1st ed. Cambridge University Press, 2008. [Online]. Available: http://books.google.com/books?id=leokXF8pLY0C&pgis=1

[7] J. Grabowski, D. Hogrefe, G. Réthy, I. Schieferdecker, A. Wiles, and C. Willcock, "An introduction to the testing and test control notation (TTCN-3)," *Computer Networks*, vol. 42, no. 3, pp. 375–403, Jun. 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128603002494

[8] C. Willcock, T. Deiß, S. Tobies, S. Keil, F. Engler, and S. Schulz, *An Introduction to TTCN-3*, second edi ed. John Wiley & Sons, 2011. [Online]. Available: http://books.google.com/books?id=ilwaMS9PbxcC&pgis=1

[9] E. M. Clarke and J. M. Wing, "Formal methods: state of the art and future directions," *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, Dec. 1996. [Online]. Available: http://dl.acm.org/citation.cfm?id=242223.242257

[10] T. Kropf, *Introduction to Formal Hardware Verification*, 1st ed. Springer-Verlag New York, Inc., 1999. [Online]. Available: http://books.google.com.co/books/about/Introduction_to_Formal_Hardware_Verifica.html?id=p3xSw3AIlToC&pgis=1

[11] B. Vlaovič, A. Vreže, Z. Brezoçnik, and T. Kapus, "Verification of an SDL Specification – a Case Study," *Electrotechnical Review*, vol. 72, pp. 14–21, 2005.

[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008. [Online]. Available: http://books.google.com/books?id=nDQiAQAAIAAJ&pgis=1http://f3.tiera.ru/2/Cs_Computerscience/CsF_Formalmethods/BaierC.,KatoenJ.PrinciplesofModelChecking(MIT,2008)(ISBN9780262026499)(O)(994s)_CsAl_.pdf

[13] M. Bozga, S. Graf, O. Ileana, O. Iulian, and S. Joseph, *Formal Methods for the Design of Real-Time Systems*, ser. Lecture Notes in Computer Science, M. Bernardo and F. Corradini, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 3185. [Online]. Available: http://www.springerlink.com/index/10.1007/b110123