Pontificia Universidad Javeriana Cali

Facultad de Ingeniería

Ingeniería Electrónica

Anteproyecto de Trabajo de Grado

Modelado, prueba y verificación de sistemas distribuidos usando RTDS e IFx

Juan Pablo Girón Ruiz

Director: Dr. Eugenio Tamura

9 de Julio de 2014

Santiago de Cali, 9 de Julio de 2014.

Señores
Pontificia Universidad Javeriana Cali.
Dra. Ana Victoria Prados
Directora Carrera de Ingeniería Electrónica.
Cali.

Cordial Saludo.

Por medio de la presente me permito informarle que el estudiante de Ingeniería Electrónica Juan Pablo Girón Ruiz (cod: 3000061050), trabaja bajo mi dirección en el proyecto de grado titulado "Modelado, prueba y verificación de sistemas distribuidos usando RTDS e IFx".

Atentamente,

Dr. Eugenio Tamura

Santiago de Cali, 9 de Julio de 2014.

Señores
Pontificia Universidad Javeriana Cali.
Dra. Ana Victoria Prados
Directora Carrera de Ingeniería Electrónica.
Cali.

Cordial Saludo.

Me permito presentar a su consideración el anteproyecto de grado titulado "Modelado, prueba y verificación de sistemas distribuidos usando RTDS e IFx" con el fin de cumplir con los requisitos exigidos por la Universidad para llevar a cabo el proyecto de grado y posteriormente optar al título de Ingeniero Electrónico.

Al firmar aquí, doy fe que entiendo y conozco las directrices para la presentación de trabajos de grado de la Facultad de Ingeniería aprobadas el 26 de Noviembre de 2009, donde se establecen los plazos y normas para el desarrollo del anteproyecto y del trabajo de grado.

Atentamente,

_____

Juan Pablo Girón Ruiz
Código: 3000061050

Índice general

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 [Sch04]. Diversas áreas de las ciencias de la computación y matemáticas 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[1] hasta el uso de métodos formales.

Garantizar que un sistema no tenga errores se ha convertido en un reto cada vez más difícil para la ingeniería y las ciencias de las matemáticas. Existen sistemas en los cuales no se admiten errores, por ejemplo en los sistemas de aviación, sistemas de control de plantas nucleares, etc. [Bow00]. Sin embargo, 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 con métodos separados es insuficiente [BBC+02].

Durante décadas se ha visto que los métodos formales (*formal methods*) y las pruebas (*Testing*) han sido rivales; en efecto, cientos de científicos defienden que verificar los modelos por medio de estructuras de lógica matemática bien definidas está por encima de hacer simples pruebas funcionales, como lo son las de caja negra. No obstante, recientemente se ha visto que los métodos formales y las pruebas son complementarios, pero lastimosamente aplicar estas técnicas para garantizar sistemas libres de errores está muy lejos de ser realidad [Hoa96, BBC+02, HKL+09].

Uno de los más famosos métodos formales usados en la industria es el Model Checking [MMT09], el cual es una técnica de verificación automática que dado un modelo y una propiedad formal determina si dicho modelo la satisface, y en caso que no pueda hacerlo es capaz de informarle al desarrollador dónde está el error para corregirlo [Ari12, MMT09].

Por otra parte, uno de los lenguajes más conocidos a nivel mundial para realizar pruebas funcionales de caja negra es *Testing and Test Control Notation Version 3*, TTCN-3, [WDT+11, ETS], el cual ha sido desarrollado y estandarizado por *European Telecommunication Standards Institute*, ETSI. TTCN-3 provee diversas características en cuanto al manejo de mensajes, niveles de abstracción, módulos para codificación y decodificación de los mensajes, entre otras [WDT+11, ETS].

El esquema que se pretende llevar a cabo en este trabajo de grado es:

- Mostrar la especificación y descripción de un sistema a partir de máquinas de estados finitas usando el lenguaje *Specification and Description Language* (SDL) y su interacción usando *Message Sequence Chart*[2](MSC).
- Una vez obtenido el modelo del sistema, mostrar cómo a partir de pruebas funcionales de tipo caja negra se puede ejercitar el modelo en los niveles más finos de abstracción.
- Para comprobar que el sistema sea confiable, se verificarán formalmente algunas de las propiedades del sistema, las cuales se consideren más críticas para su funcionamiento.
- Para ejecutar los anteriores pasos, se tomará como caso de estudio el sistema de parqueaderos de automóviles de la Pontificia Universidad Javeriana Cali; éste caso será útil para mostrar que el trabajo en conjunto de pruebas y verificaciones formales, posibilita implementar sistemas más confiables.

[1] Prueba de caja negra: Es una prueba funcional que consiste en estimular el sistema a probar con las entradas apropiadas que debe recibir la aplicación, y revisar si las salidas son las deseadas; lo anterior se lleva a cabo sin tener conocimiento de la estructura/funcionamiento interno del sistema.

[2] Más información de MSC referirse a la recomendación Z.120 disponible en: https://www.itu.int/rec/T-REC-Z.120-201102-I/en

Para la descripción y especificación del sistema se hará uso de SDL y MSC, para las pruebas funcionales de caja negra se utilizará TTCN-3, y finalmente para las verificaciones formales se hará uso de IFx[3], la cual es una herramienta desarrollada por VERIMAG[4] que por medio del acceso al árbol abstracto desde una especificación de SDL la traslada a un lenguaje intermedio llamado IF, y a través de este último se puede verificar usando un algoritmo de *Model Checking*[5], que ya viene integrado en la herramienta. Un factor diferenciador que se le quiere dar al trabajo de grado es poder hacer uso de los tres lenguajes mencionados anteriormente con un único software que se llama *Real Time Developer Studio*[6], RTDS, software desarrollado por la compañía Francesa *Pragmadev*.

[3] Más Información de la herramienta disponible en: `http://www-if.imag.fr`

[4] Para mas información acerca de VERIMAG consultar la siguiente página: `http://www-verimag.imag.fr/?lang=en`

[5] Model checking es una técnica de verificación que, dado el modelo del sistema bajo estudio y la propiedad requerida, permite decidir automáticamente si la propiedad es satisfecha o no [CW96, Ham09]

[6] Más Información del proveedor del software disponible en:`http://www.pragmadev.com`

Capítulo 1

DESCRIPCIÓN DEL PROBLEMA

1.1. Planteamiento del Problema

La complejidad de los sistemas de Software/Hardware se ha venido incrementando significativamente debido a las necesidades de la industria [CW96, Sch04, Ser12]; debido a estas exigencias, la confiabilidad de dichos sistemas es un tema que está siendo abordado desde diversos campos de las ciencias de la computación y las matemáticas. Es necesario garantizar que los modelos a implementar satisfagan propiedades de funcionamiento y sobre todo que sean seguros en situaciones donde la vida humana esté en juego [Hoa96, Bow00], como por ejemplo: en los sistemas involucrados en el campo de la aviación, y otros medios de transporte, etc.

Actualmente, entre las más conocidas técnicas que existen para la detección y depuración de errores, se encuentran las pruebas funcionales a sistemas y el uso de métodos formales; sin embargo, a pesar de su existencia, es cada vez más complejo garantizar sistemas sin errores haciendo uso de una sola técnica [Hoa96]. Por ejemplo, los métodos formales se fundamentan en estructuras de lógica matemática bien definidas que no son tan sencillas de usar en diferentes etapas del desarrollo del sistema, debido a que exigen una alta demanda de tiempo y contar con personal altamente entrenado [Ser12]; por otra parte, tampoco sería conveniente usar sólo pruebas funcionales a sistemas de seguridad crítica pretendiendo garantizar el correcto funcionamiento al cien por ciento.

Por lo anterior, dado el incremento acelerado tanto en la complejidad como en la aplicación de los sistemas embebidos, es necesario proveer sistemas con niveles de confiabilidad suficiente para su correcto funcionamiento, haciendo uso combinado de dos técnicas en una sola metodología para la detección de errores en modelos a implementar, en este caso se trata de los ya mencionados Métodos formales y Pruebas funcionales.

1.1.1. Formulación

¿Es posible minimizar el número de errores en un sistema, usando en conjunto técnicas de métodos formales y pruebas sobre diferentes etapas de desarrollo del sistema?

1.1.2. Sistematización

1. ¿Cuál lenguaje semi-formal permite expresar un sistema a través de máquinas de estados finitas?
2. ¿Por qué es necesario representar los sistemas a través de lenguajes formales?
3. ¿Cómo diferenciar en qué etapas del desarrollo de un sistema es conveniente aplicar métodos formales o pruebas?
4. ¿Qué tipo de técnicas son apropiadas para verificar si un sistema satisface unas condiciones dadas?
5. ¿Cómo usar estructuras de lógica matemática para verificar modelos?

1.1.3. Posible título del trabajo de grado

Para definir el título del trabajo de grado se han especificado las fases y las herramientas a usar en la verificación del correcto funcionamiento de un sistema distribuido, el cual empieza desde su descripción en un lenguaje que no tenga ambigüedades y que se pueda manipular fácilmente para optimizar, probar y verificar a través de métodos formales. Por lo anterior se definió el siguiente título:

"MODELADO, PRUEBA Y VERIFICACIóN DE SISTEMAS DISTRIBUIDOS USANDO RTDS e IFx"

1.2. Objetivos

1.2.1. Objetivo general

Mostrar 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.

1.2.2. Objetivos Específicos

1. Comprender el correcto funcionamiento de la herramienta Real Time Developer Studio.
2. Implementar el caso de estudio usando un lenguaje que no introduzca errores por ambigüedades en la descripción, el cual posteriormente será verificado por medio de métodos formales y pruebas.
3. Determinar en qué etapas del desarrollo de sistemas es pertinente usar métodos formales o pruebas.
4. Implementar una batería de pruebas incrementales que sea apropiada con el modelo del caso de estudio y compatible con el lenguaje en el cual ha sido expresado.
5. Entender el funcionamiento de IFx para verificar formalmente el modelo del caso de estudio.
6. Definir las propiedades críticas del sistema que se desea verificar.
7. Seleccionar los tipos de observadores para la especificación de la propiedad a verificar.
8. Seleccionar las propiedades a verificar.
9. Implementar los observadores.

1.3. Justificación

Tanto la verificación formal como las pruebas funcionales son metodologías que ayudan a la detección y corrección de errores de modelos que se van a implementar. A lo largo del tiempo encontramos en la literatura que éstas dos técnicas han sido rivales; sin embargo, recientemente se puede apreciar que las ciencias computacionales y las matemáticas proponen que la mejor forma de verificar si un modelo satisface unas condiciones dadas, implica necesariamente usar estas dos técnicas en conjunto [Hoa96].

En el año 2001 en la Universidad de Brunel, Reino Unido, se creó una red llamada Métodos Formales y Pruebas, conocida por sus siglas en inglés como FORTEST que significa Formal Methods and Testing. Esta red fue financiada por el Consejo de Investigación de Ciencias Físicas e Ingeniería, por sus siglas en inglés

EPSRC[1]. El enfoque principal de FORTEST era explorar caminos que permitieran mostrar que los métodos formales y el software de pruebas son complementarios, y así facilitar los enfoques y técnicas que permitan producir sistemas de alta calidad[2].

Por lo tanto, el presente trabajo de grado pretende mostrar a lo largo de su desarrollo que es posible diseñar sistemas confiables usando tanto las pruebas como los métodos formales para la detección y corrección de errores, basándose en las investigaciones de la red FORTEST.

1.4. Alcances y Limitaciones

- El proceso de modelado del caso de estudio se hará por medio de un lenguaje que no posea ambigüedades además de soportarlo la herramienta Real Time Developer Studio, en este caso será SDL.
- El modelo del caso de estudio se limitará a especificar de manera general los procesos involucrados; algunos de ellos se asumirán como subsistemas, los cuales proporcionando una entrada retornan una salida, pero no se modelará su comportamiento dado que no es de interés del presente trabajo de grado desarrollarlo. Ej.: modelar la base de datos de usuarios del parqueadero.
- Las pruebas realizadas al modelo se harán en las primeras fases del desarrollo del mismo sin usar componentes paralelos de pruebas (PTCs) usando SDL, MSC y TTCN3.
- Las verificaciones formales no se harán en el nivel de sistema, sino sobre módulos específicos.
- Se hará uso de la herramienta SDL2IF para la transformación del modelo descrito en SDL a IF, con el fin de verificarlo formalmente.
- Los procesos de modelado y pruebas se harán usando la herramienta Real Time Developer Studio, y la parte de verificación formal se hará por medio de la herramienta IFx.

[1] Sitio Web de EPSRC: http://www.epsrc.ac.uk/Pages/default.aspx
[2] Para mas detalles: http://gow.epsrc.ac.uk/NGBOViewGrant.aspx?GrantRef=GR/R43150/01

Capítulo 2

MARCO DE REFERENCIA

2.1. Marco Teórico

En esta subsección se introducen conceptos pertinentes al desarrollo del trabajo de grado, además se ha subdividido en tres grandes partes: La primera corresponde a los conceptos del Modelado de sistemas; en este caso se hace referencia a la estructura y características que posee Specification and Description Language (lenguaje de especificación y descripción de sistemas), conocido por sus siglas en inglés SDL. En la segunda parte se definen algunos términos de las pruebas funcionales de caja negra, y se enfatiza en la estructura y característica que brinda el lenguaje Testing and Test Control Notation version 3 conocido por sus siglas como TTCN-3 para dicha fase; por último, en la tercera parte se definen algunos conceptos referentes a la verificación formal de sistemas y se hace énfasis en la herramienta que se va a usar, IFx, definiendo sus características.

2.1.1. Lenguaje para la Especificación y Descripción de sistemas distribuidos

El uso de un modelo o especificación formal elimina ambigüedades que disminuyen los errores dentro del desarrollo del sistema [HKL+09]. Especificar un sistema por medio de lenguajes formales tiene muchas ventajas entre las cuales encontramos:

- Obtener un modelo que tenga una especificación y descripción clara y concisa del sistema.
- Es posible automatizar la fase de pruebas, con el fin de hacerlas más dinámicas buscando el correcto funcionamiento del sistema.
- Los sistemas son expresados bajo lenguajes basados en matemáticas, lo que ayuda a la construcción de sistemas de alta calidad [HKL+09].
- Es posible usar verificaciones formales para la detección y corrección de errores.
- Permite ahorrar mucho tiempo/esfuerzo determinando dónde se originan los errores del sistema.

Dentro de los lenguajes que permiten expresar sistemas a partir de su especificación y descripción sin ambigüedad, se encuentra SDL, el cual posee unas características atractivas para representar sistemas reactivos[1], entre las cuales se destaca representar sistemas con múltiples agentes a través de máquinas de estados finitas extendidas.

A continuación se definirán algunas estructuras que posee dicho lenguaje, definidas por la recomendación Z.100 de la ITU-T [IT02]:

[1] Sistemas reactivos: Son aquellos sistemas que generan una salida a partir de un estímulo externo que proviene del ambiente.

2.1.1.1. Definiciones en SDL

- Agent (Agente): Es usado para denotar un sistema, bloque o proceso que contiene una o más máquinas de estados finitas extendidas.
- Block (Bloque): Un bloque es un agente que puede contener uno o más bloques o procesos paralelos, y podría también contener una o varias máquinas de estados finitos extendidas que poseen y manejan datos dentro del bloque.
- Channel (Canal): Un canal es un camino de comunicación entre dos agentes.
- Environment (Ambiente): El ambiente es el entorno en el que se desempeña el sistema.
- El tipo PID es usado cuando los elementos de datos son referenciados a un agente. Existen cuatro variables de tipo PID bien definidas en SDL las cuales son:
  - SELF: Es el identificador de proceso de la instancia actual.
  - SENDER: Es el identificador de proceso que envió la última señal.
  - PARENT: Es el identificador de proceso que creó la última instancia.
  - OFFSPRING: Es el identificador de proceso de la última instancia creada.
- Procedure (Procedimiento): Un procedimiento es una encapsulación de una parte del comportamiento de un agente, que puede ser usado en diferentes partes dentro del mismo. Otros agentes puedan hacer llamados a un procedimiento remoto.
- Signal (Señal): La principal manera de comunicación es por medio de señales que son salidas del agente emisor y entradas al agente receptor.
- State (Estado): Una máquina de estados finita extendida de un agente está en un estado si este está esperando por un estímulo.
- Stimulus (Estímulo): Un estímulo es un evento que puede causar que un agente que está en un estado ejecute una transición.
- System (Sistema): Un sistema es el agente más exterior que se comunica con el ambiente.
- Timer (Temporizador): Un temporizador es un objeto de propiedad de un agente que causa una señal de estímulo cuando éste alcanza un determinado tiempo.
- Transition (Transición): Una transición es una secuencia de acciones que un agente realiza hasta que éste ingresa a un estado.
- Type/Sort (Tipo): Un tipo es un conjunto de elementos que posee características en común cuya definición puede ser usada para la creación de otras instancias; también se pueden formar otros tipos.
- Value (Valor): El término valor es usado para la clase de datos que son accesados directamente. Los valores pueden ser pasados con libertad entre agentes.

2.1.1.2. Notación de SDL

En SDL existen dos notaciones las cuales son:

- SDL/GR Representación gráfica de SDL (En inglés: Graphic Representation form of SDL)
  - Provee elementos gráficos para los conceptos más importantes del lenguaje.
  - Tiene una notación textual para aquellos elementos que no es apropiado representarlos gráficamente.
- SDL/PR Representación textual de SDL (En inglés: Textual Phrase Representation form of SDL)
  - Usado principalmente para el desarrollo de compiladores.

La recomendación de la ITU-T Z.100 enfatiza en la representación gráfica de SDL. Además el software a usar en la fase de modelado del caso de estudio brinda la posibilidad de usar SDL/GR.
La Figura 2.1 es un ejemplo para mostrar la diferencia entre las notaciones SDL/GR y SDL/PR:

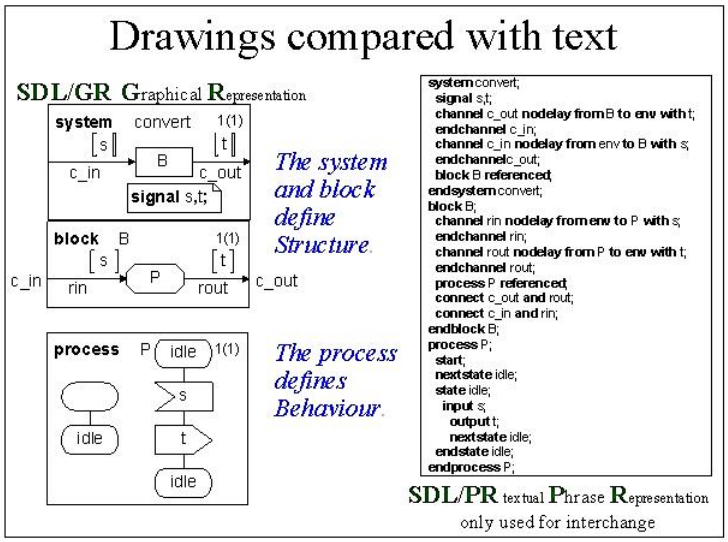

Figura 2.1: Ejemplo comparación representación gráfica y textual en SDL. Fuente http://www.sdl-forum.org/sdl2000present/sld006.htm

A partir de este momento los ejemplos relacionados con el modelado de sistemas se harán por medio de componentes gráficos.

2.1.1.3. Requerimientos de especificación usando Message Sequence Charts (MSC)

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)[2] es un lenguaje textual y gráfico que permite expresar requerimientos de especificaciones, simulación y validación, además de describir el comportamiento de comunicación entre las

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

entidades del sistema y el ambiente [Ebn04]. Actualmente existe una relación directa entre la especificación realizada en un diagrama MSC y SDL, que facilita la parte de diseño del modelo.

Las Figuras 2.2 y 2.3 muestran un ejemplo de la relación MSC-SDL.

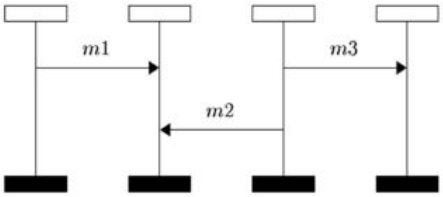

Figura 2.2: Diagrama MSC interacción entre cuatro procesos

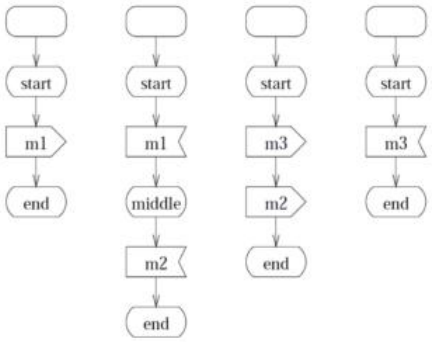

Figura 2.3: Diseño en SDL de la especificación en MSC

2.1.2. Software de Pruebas

2.1.2.1. Terminología 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 [Hom13].

De acuerdo al estándar ANSI/IEEE 1059 una prueba se puede definir como: Un proceso de análisis de un elemento de software para detectar las diferencias entre las condiciones existentes y requeridas (es decir defectos / errores / fallos) y para evaluar las características del elemento software [IEE94].

Las pruebas pueden ser estáticas o dinámicas. Las primeras, prueban el componente o el sistema a nivel de especificación o de implementación sin tener que ejecutar éste [AO08, Hom13, ISO13]. En las pruebas dinámicas, el componente o sistema está en ejecución y se estimula con entradas reales [AO08, Hom13].

La técnica de diseño de pruebas dinámicas permite identificar las condiciones de la prueba; adicionalmente ésta se clasifica dentro de tres categorías, basadas en cómo son derivadas las entradas de las pruebas [ISO13]. Esas categorías son: Basada en la especificación, que consiste en proveer los casos de prueba desde una base de prueba describiendo el comportamiento esperado del elemento a probar [ISO13]; la segunda categoría está basada en la estructura, que consiste en derivar los casos de prueba desde una característica estructural, por ejemplo la estructura de un código fuente [ISO13]. Finalmente la tercera categoría corresponde a las pruebas basadas en la experiencia, que se basa en un conocimiento previo, bien sea en un sistema en particular o en métricas de proyectos previos [Hom13].

Mayoritariamente se emplean dos métodos de pruebas que son: Caja Negra (En inglés: Black-Box) y Caja blanca (En inglés: White-Box). El método de caja negra está basado en los requerimientos y especificaciones para determinar si el modelo es correcto o no [Hom13, AO08]; una característica importante de éste método es que no es necesario conocer el sistema interno. Por el contrario, el método de caja blanca hace parte de la categoría basada en la estructura: el caso de prueba se deriva desde el código fuente interno del modelo [AO08]; la hipótesis fundamental consiste en que el modelo cumple con los requerimientos del cliente.

Los tipos de integración de pruebas son principalmente dos: De abajo hacia arriba (Bottom-up) y de arriba hacia abajo (Top-Down). La primera consiste en diseñar pruebas que empiecen desde los niveles más finos del sistema a los niveles más altos; la integración Top-Down consiste en diseñar pruebas desde los niveles más altos del sistema a los más finos.

2.1.2.2. TTCN-3 (Testing and Test Control Notation Version 3)

Para minimizar errores en los sistemas es necesario usar diferentes técnicas que permitan cumplir esta meta; por ejemplo, es útil realizar las pruebas funcionales de caja negra para probar el correcto funcionamiento en las primeras etapas del desarrollo del sistema. El presente trabajo de grado pretende usar las pruebas funcionales de caja negra en algunos módulos del caso de estudio haciendo uso del lenguaje TTCN-3. Parte del modelo arquitectónico de TTCN-3 está soportado por la herramienta RTDS.

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 [GHR+03]. TTCN-3 se ha construido desde un núcleo de lenguaje textual que posibilita la interacción con otros lenguajes de descripción, por ejemplo SDL [GHR+03, WDT+11]. Una de las características que presenta TTCN-3 es que su sintaxis textual es similar a lenguajes típicos de programación, por ejemplo: C, C++ o Java.

2.1.2.3. Conceptos básicos de TTCN-3

- System Under Test SUT (Sistema Bajo Prueba): Es el sistema el cual se va a someter a pruebas.
- Module (Módulo): Es donde está recopilado el código TTCN-3 y se encuentra conformado por: Una parte de definiciones y una parte de control [WDT+11].
- Module definitions part (Parte de definiciones): Especifica las definiciones en el nivel superior del módulo; éstas pueden ser usadas en cualquier parte incluso en la parte de control.
- Module control part (Parte de control): Es la parte principal del programa de TTCN-3; en esta parte se describe la secuencia de la ejecución de los casos de prueba (Test cases).
- Test Cases (Casos de Prueba): Los casos de prueba son especificados en la parte de las definiciones del módulo y son llamados en la parte de control. Un caso de prueba define el comportamiento que es ejecutado con el fin de observar si el sistema pasa la prueba o no [GHR+03].
- Test System (Sistema de Prueba): Ejecuta un caso de prueba y está conformado por un conjunto de componentes de prueba interconectados con unos puertos de comunicación bien definidos y una explícita interfaz de sistema de prueba (en inglés: Test System Interface, TSI) [GHR+03].
- Test Component (Componente de prueba): Ejecuta los casos de prueba (Test cases) [GHR+03].
- Main Test Component MTC (Componente de prueba principal): La función principal es determinar a través de los resultados de los componentes de prueba si ésta pasa o no.
- Verdict (Veredicto): Es una variable implícita que corresponde al resultado de la prueba.

- Matching mechanism (Mecanismo de Coincidencia): TTCN-3 tiene integrado un mecanismo de coincidencia que permite evaluar si el sistema satisface ciertas condiciones, definidas en los casos de prueba. En TTCN-3 encontramos los siguientes valores para el veredicto:
  - Pass: Significa que el SUT se comportó de acuerdo al propósito de la prueba.
  - Inconc: Significa que no se puede determinar si el SUT pasó o falló la prueba.
  - Fail: 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.
  - None: Es el valor inicial, cuando el veredicto no ha sido asignado.
- Port (Puerto): Es el lugar donde los mensajes llegan o salen al componente de prueba. Un puerto es modelado como una cola infinita tipo FIFO.
- Template (Plantilla): Son entidades en TTCN-3 usadas para describir el contenido de operaciones de comunicación. Poseen mecanismos de coincidencia para los mensajes de llegada y se omite en los mensajes de salida.
- Alt-statement (Alt-Declaración): Permite definir distintas operaciones de bloqueo, que permiten manejar los mensajes entrantes y determinar si la prueba pasa o no.
- Comunicación: Se refiere al intercambio de señales entre componentes o con el SUT. La comunicación pueden ser basada en mensajes o basada en procedimientos.
- Comunicación basada en mensajes: Se intercambian mensajes con el SUT a través de dos operaciones: send y receive. La operación send transmite un mensaje al SUT a través de un puerto específico. La operación receive es una operación de bloqueo que tiene como finalidad comparar el mensaje entrante y procede a determinar si coincide con el esperado,para determinar si la prueba pasa o no.

2.1.2.4. Modelo arquitectónico del Sistema de Prueba en TTCN-3

La Figura 2.4 representa la arquitectura general del lenguaje de TTCN-3. De manera general los adaptadores en TTCN-3 (TTCN-3 Adapters) están conformados por un adaptador de sistema y uno de plataforma que sirven para implementar funciones externas y adaptar los mensajes al SUT. Por otra parte en la capa del ejecutable de TTCN-3 (TTCN-3 Executable) se encuentran componentes de manejo (Component Handling), codificación y decodificación (Codecs), y un ejecutable de TTCN-3. Todo esto sirve para: La distribución y comunicación entre componentes de prueba en paralelo, la transformación de mensajes a un formato entendible por TTCN-3, y la representación del comportamiento específico en el nivel de TTCN-3, respectivamente. Finalmente en el Control y Administración de pruebas (Test Management and Control) se encuentran un Administrador de pruebas (Test Management) y una parte para el registro de eventos de prueba (Test Logging) que sirven para: proveer control sobre el orden de ejecución de los casos de pruebas, y para el manejo de registro de sistema de prueba, respectivamente.

La herramienta RTDS soporta parcialmente la interfaz de control de TTCN-3 (TCI). Adicionalmente la interfaz TSI (Test System Interface) es proporcionada por la herramienta cuando la prueba se va hacer sobre un sistema descrito en SDL. El diseño de prueba del caso de estudio se hará de forma secuencial para los módulos más finos del sistema, con el objetivo de garantizar su correcto funcionamiento; por lo tanto, no es de interés en este trabajo de grado implementar componentes paralelos de pruebas, lo que implica no hacer uso del componente de manejo (Component Handling). Adicionalmente, se aprovechará que la herramienta RTDS proporciona implícitamente una interfaz para la comunicación con un SUT descrito en SDL; por lo tanto, tampoco es de interés hacer uso de adaptadores para la realización de pruebas al sistema.

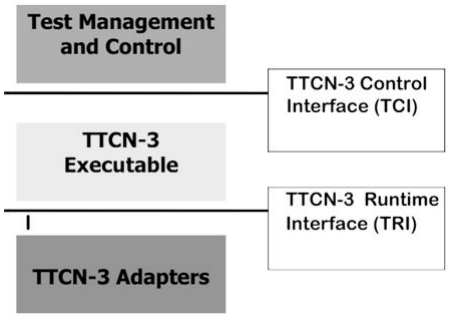

Figura 2.4: Modelo arquitectónico simplificado de TTCN-3. Fuente [AR11]

2.1.3. Verificaciones Formales

2.1.3.1. Métodos Formales

Los métodos formales según [CW96] son lenguajes, técnicas y herramientas basados en estructuras matemáticas con el objetivo de especificar y verificar sistemas. El uso de formalismos en la verificación de sistemas no garantiza que estos estén libres de errores, pero brinda la posibilidad de expresar modelos sin ambigüedades y con un mejor entendimiento.

Dentro de los Métodos formales se encuentra la especificación formal de sistemas que consiste en expresar por medio de estructuras de lógica matemática las propiedades deseadas, eliminando ambigüedades que inducen errores en el diseño [CW96]. La especificación formal es una descripción clara y concisa del comportamiento de alto nivel y/o de las propiedades del sistema [Kro99]. Por otra parte los métodos formales también son usados para verificar si el sistema modelado satisface ciertas propiedades, que en muchos casos deben de cumplir los sistemas críticos [CW96].

2.1.3.2. Verificación Formal de Hardware

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) [Kro99]. 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 no es tan sencillo lograr estimular el circuito o el sistema con todas las posibles entradas que va a tener durante su funcionamiento.

Como se muestra en el flujo de diseño usando verificación en la Figura 2.5, encontramos que para la verificación de sistemas es necesario que el sistema sea especificado de manera rigurosa (System specification). Una vez definido el sistema a través de lenguajes que no permitan ambigüedades en la descripción, se definen las propiedades que se desea verificar. Paralelo a la definición de las propiedades se puede efectuar el proceso de diseño (Design Process) hasta obtener un producto o prototipo (product or prototype) del sistema deseado. Finalmente, se verifica a través de métodos formales que el prototipo satisface las propiedades definidas anteriormente, dando como resultado un éxito o fallas por medio de contraejemplos [VVBK05], como lo hace la técnica del Model Checking.

El flujo de diseño usando verificación mostrado anteriormente no está diseñado estrictamente para sistemas Hardware, sino que también puede ser usado en desarrollo de Hardware/Software.

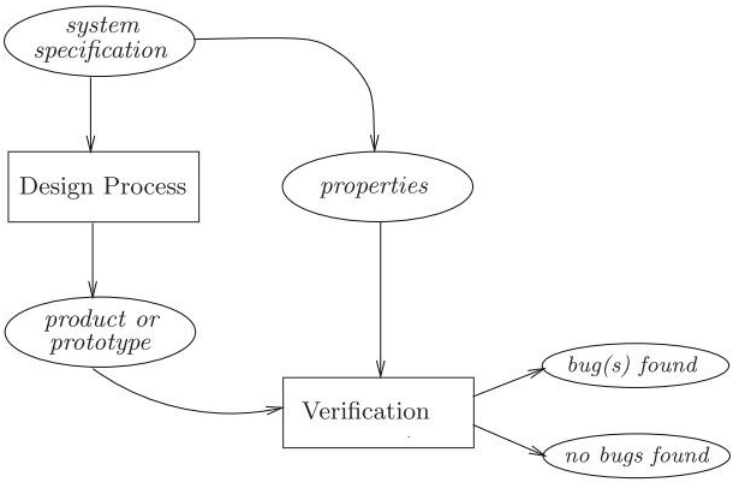

Figura 2.5: Vista general de un sistema de verificación. Fuente: [BK08, p. 3]

2.1.3.3. Model Checking

El Model Checking es una técnica automática [VVBK05] 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 [CW96]. La exploración de todos los posibles estados del sistema se da de forma exhaustiva.

El Model Checking 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 [BK08, VVBK05].

A continuación se describe el proceso del Model Checking de una forma general [BK08]:

- Fase del Modelado (Modelling Phase):
  - Modelar el sistema por medio de un lenguaje que el Model Checker pueda manejar.
  - Evaluar rápidamente el modelo a través de simulaciones para eliminar fallas básicas del modelo.
  - Especificar la propiedad que se desea verificar en el modelo por medio de un lenguaje de especificación.
- Fase de ejecución (Running phase): En esta fase se ejecuta el model checker para verificar si el modelo satisface la propiedad.
- Fase de análisis (Analysis phase):
  - Si se cumplió la propiedad se continúa verificando las otras en caso de haber más.
  - Si la propiedad no se cumplió:
    1. Analizar el modelo con el contraejemplo generado por el model checker.
    2. Refinar el modelo, diseño o propiedad.
    3. Repetir el procedimiento para verificar nuevamente la propiedad.
  - Si se quedó corto de memoria: Intentar reducir el modelo e intentar nuevamente.

2.1.3.4. 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 [BGI+04].

La herramienta IFx posee características que proveen grandes ventajas a los diseñadores de sistemas tales como:

- Soporta modelado de alto nivel con formalismos tales como SDL y UML.
- Permite trasladar modelos de alto nivel a una interpretación intermedia conocida como IF.
- El modelo semántico de IF posee una representación rica y suficientemente expresiva para permitir describir los conceptos básicos y estructura del lenguaje fuente.
- IF es una representación intermedia basada en un Autómata temporizado extendido, con variables de datos discretas, comunicación primitiva y creación y destrucción dinámica de procesos.
- IF permite la simplificación de modelos, lo que permite optimizarlo para poder ser verificado por medio de la técnica Model Checking.
- La propiedad que se desea verificar se puede expresar usando un observador (observer) que se puede clasificar como: pure (puro), cut (Poda) o intrusive (intrusivo).

La Figura 2.6 representa la arquitectura IFx[3]:

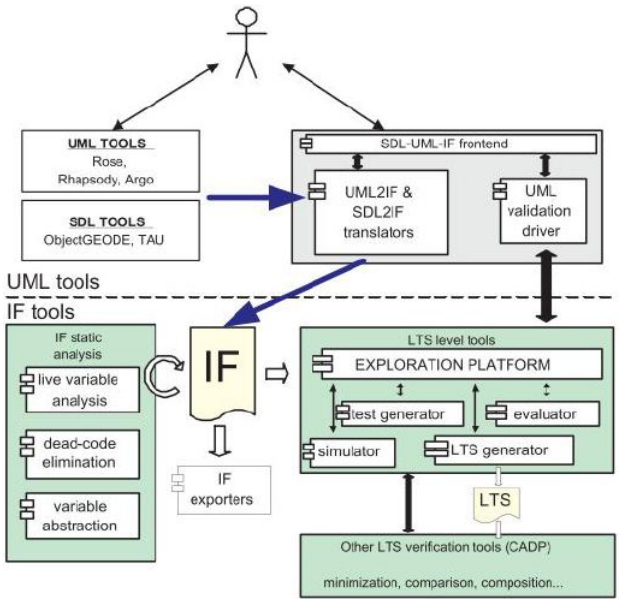

Figura 2.6: Arquitectura de la Herramienta IFx. Tomada de [BGI+04]

El propósito de este trabajo de grado es hacer uso de esta herramienta y no es de interés mostrar la interacción de cada componente de dicha herramienta. Para saber más respecto de su estructura y semántica referirse a [BGI+04].

[3]Página oficial de la herramienta IF: http://www-if.imag.fr/index.html

2.1.3.5. Conceptos básicos de IFx

- Process (Proceso): Un proceso describe el comportamiento secuencial incluyendo transformación de datos, comunicación y creación de procesos. Un proceso se define como un autómata temporizado, que tiene un identificador único conocido como pid y una memoria local consistiendo de: variables, control de estados y una cola de mensajes que han llegado y no han sido consumidos. Para cada instancia de un proceso en SDL la herramienta IFx crea un proceso IF.

- Variables (variables): Cada variable/timer que se declara en SDL es traducido al correspondiente proceso IF.

- States (Estados): Todos los estados del modelo en SDL, incluidos los de start y stop, son traducidos dentro de un control de estados estable de IF (stable IF Control States). Cada decisión en SDL se traduce en un estado inestable (non stable IF state).

- Transitions (Transiciones): Las transiciones definen el camino entre dos estados de control de IF; pueden ser de tres tipos: eager (impaciente) aquí las transiciones tienen absoluta prioridad sobre el progreso del tiempo, delayable (retardable) pueden permitir el progreso de tiempo mientras está habilitada esta transición, lazy (perezoso) no evita el progreso del tiempo.

- Observers (Observadores): Los observadores permiten definir la propiedad deseada a verificar. En IF podemos definir tres tipos de observadores los cuales son [BGI+04]:

  - Pure (Puro): Expresan los requerimientos para ser verificados en el sistema.
  - Cut (Poda): Adicionalmente al observador pure, este permite guiar la simulación hacia un determinado camino de ejecución, ayudando a restringir el comportamiento del ambiente.
  - Intrusive (Intrusivo): Este observador es el más completo de todos, dado que permite manipular el sistema enviando señales y cambiando variables.

2.1.4. Trabajos Relacionados

La técnica del Model Checking ha sido usada desde comienzos de los años 90 para la verificación de propiedades en Hardware. Algunos de los casos más notables en este proceso han sido los siguientes proyectos que se puede encontrar en [CW96, WLBF09]. En [CW96] mencionan el proyecto IEEE Futurebus+ como caso de estudio para mostrar las ventajas de la técnica del Model Checking, usando una herramienta de verificación automática por primera vez. Este proyecto consistía en verificar la coherencia de la cache descrita en el estándar 896.1-1991 de la IEEE Futurebus+. Este proyecto fue realizado por Clarke y sus estudiantes de la Universidad de Carnegie Mellon. En [CW96] se pueden encontrar diferentes proyectos tanto en Hardware como en Software donde se usan los métodos formales para garantizar que el sistema cumpla con propiedades críticas; se puede encontrar además que el proyecto llamado NewCore fue el primero en ser verificado formalmente en su totalidad, éste se describía en 7.500 líneas de código en SDL, excluyendo comentarios, que fueron verificados, hallando 112 errores que fueron corregidos.

Compañías como Intel mencionan que el uso de los métodos formales para la verificación de hardware ha sido útil para garantizar sistemas confiables y de alta calidad, además de permitir mejorar los procesos de desarrollo en sus primeras etapas del sistema [Fix08]. Recientemente en [WLBF09] se puede encontrar un estado del arte actualizado de las experiencias que han obtenido las industrias en el uso de los métodos formales. La recolección de información se hizo por medio de un cuestionario enviado a las personas que han estado involucradas en el uso de los métodos formales en la construcción de sistemas. La recopilación de información se dio entre Noviembre del 2007 a Diciembre del 2008; a través de esta encuesta se puede

apreciar que la aplicabilidad de los métodos formales no sólo es en el campo de la ingeniería, sino también, en las finanzas, Salud, Defensa, entre otros.

Los grupos de investigación de VERIMAG, que es un centro de investigación líder en sistemas embebidos, han propuesto un lenguaje intermedio para traducir modelos escritos en SDL y UML para verificar los mismos usando la técnica de Model Checking. En la literatura encontramos distintos trabajos de grado [BP06, PT06] que han probado la herramienta de IFx para verificar que algunas propiedades se mantienen en el modelo. En [BP06] usaron IFx sobre un sistema de telefonía móvil, como caso de estudio, donde se verificaron algunas propiedades como: La consistencia del tipo de llamada que asigna el sistema con el tipo de llamada que obtenga el usuario. En [PT06] hacen uso de la herramienta IFx para verificar un controlador de semáforos que está descrito en UML, la propiedad que desean probar es: "Se busca sí en un momento dado dos semáforos están en verde ó uno en verde y otro en ámbar", lo anterior representa un estado no deseado del sistema, lo que se concluye es que a través de la herramienta IFx se pudo determinar la falla del modelo.

En [VVBK05] hacen uso de los métodos formales para verificar propiedades en un caso de estudio referente a redes de comunicación. El modelado del sistema se hizo por medio de SDL y la verificación formal fue a través de la técnica de Model Checking; se tradujo el modelo de SDL a IF por medio de la herramienta SDL2IF, con el fin de obtener un sistema en un lenguaje al que se le puede aplicar algún model checker.

En [GJ01] plantean y desarrollan un experimento para la verificación de la capa de control dinámica del protocolo MASCARA (Mobile Access Scheme based on Contention And Reservation for ATM). Hacen uso de diferentes técnicas de reducción del sistema, con el objetivo de optimizar el modelo y por ende el tiempo de respuesta usando la técnica de Model Checking; para más información de las técnicas usadas ver [GJ01]. La forma de diseñar las propiedades a verificar fue de manera incremental, donde se empezaba con propiedades sencillas y se terminó verificando la capa de control dinámica del protocolo MASCARA. Para ver los resultados puede referirse a [GJ01].

En [Mar03] se escogió un bloque del Software de enrutamiento telefónico desarrollado en Alcatel Network Systems Romania, para verificar algunas propiedades. Dicho bloque es el encargado de realizar el tipo de diálogo apropiado dependiendo de los parámetros de los mensajes de inicio y subsecuentes. El bloque cuenta con dos interfaces: una upstream para el que llama y una downstream para el servidor. En este caso de estudio hacen uso de IF como lenguaje intermedio para aplicar la técnica de Model Checking y verificar básicamente cuatro propiedades las cuales son: DeadLocks, Progeso, Abortar y Cierre.

Con respecto a las pruebas funcionales de caja negra usando el lenguaje TTCN-3 encontramos estudios referente al diseño de casos de pruebas a partir de modelos expresados por medio de SDL y MSC. En [Ebn04] proponen una traducción de elementos de MSC a lenguaje de pruebas TTCN-3, en este paper definen una nueva semántica de MSC para permitir distintas especificaciones de casos de prueba en TTCN-3 que pueden ser concurrentes o no. En [WDT+11] hay una breve descripción de cómo el lenguaje TTCN-3 ha sido usado para diseñar pruebas al último estándar de sistemas de comunicaciones móviles conocido como LTE. Se destacan los campos de acción en los cuales el lenguaje TTCN-3 está siendo usado, entre estos encontramos [Sch10]: sector automovilístico, equipos médicos, distribución y transmisión de potencia, Finanzas, Aviación, Ferrocarriles.

Este trabajo de grado pretende hacer uso de una metodología que combina dos técnicas, que son los métodos formales y las pruebas, para la minimización de errores en el diseño de sistemas descritos en SDL y pretende verificar algunas propiedades de algunos de sus módulos haciendo uso de la técnica de Model Checking. Adicionalmente, se desea hacer pruebas funcionales de caja negra por medio del uso del lenguaje TTCN-3 en las primeras etapas del desarrollo del sistema.

2.2. Metodología de la investigación

El tipo de estudio en el que se enmarca este proyecto es de tipo explicativo y descriptivo. La primera es porque el proyecto de grado trata de aplicar dos técnicas en una metodología que combina, métodos formales y pruebas funcionales de manera conjunta para minimizar los errores en los sistemas. El aspecto descriptivo está involucrado en mostrar la aplicación de ambas técnicas sobre un caso de estudio y haciendo uso de dos herramientas que son: RTDS para el modelado y pruebas e IFx para la verificación formal.

2.3. Resultados esperados

Se espera mostrar a través de un caso de estudio, que es posible minimizar los errores en el diseño de éste usando la combinación de los métodos formales y Pruebas sobre distintas etapas del diseño del caso de estudio.

2.4. Descripción de recursos

2.4.1. Director

- Eugenio Tamura Morimitsu.
  Dr. en Arquitectura y Tecnología de los Sistemas Informáticos.

2.4.2. Recursos Técnicos

- Computador personal
- Material bibliográfico (Revistas, libros)
- Papelería (Impresiones)
- Conexión a internet
- Herramienta Real Time Developer Studio
- Herramienta IFx

2.4.3. Presupuesto

Para llevar a cabo el trabajo de grado planteado se ha realizado la siguiente tabla detallando el valor de dicho proyecto, considerando que la duración para su realización es de 28 semanas:

Detalle	**Costo por tiempo**	**Costo**
Investigador	480.000/semana	13'440.000
Computador Personal	25.000/semana	700.000
Material Bibliográfico	500.000/Investigación	500.000
Papelería	120.000/Investigación	120.000
Conexión a Internet	15.000/semana	420.000
	Total	15'180.000

Cuadro 2.1: Presupuesto

2.5. Tabla de Contenido

2.6. Cronograma

En la Figura 2.7 se observa el cronograma de actividades por semanas. La reunión con el director del proyecto de grado se realizará cada 15 días.

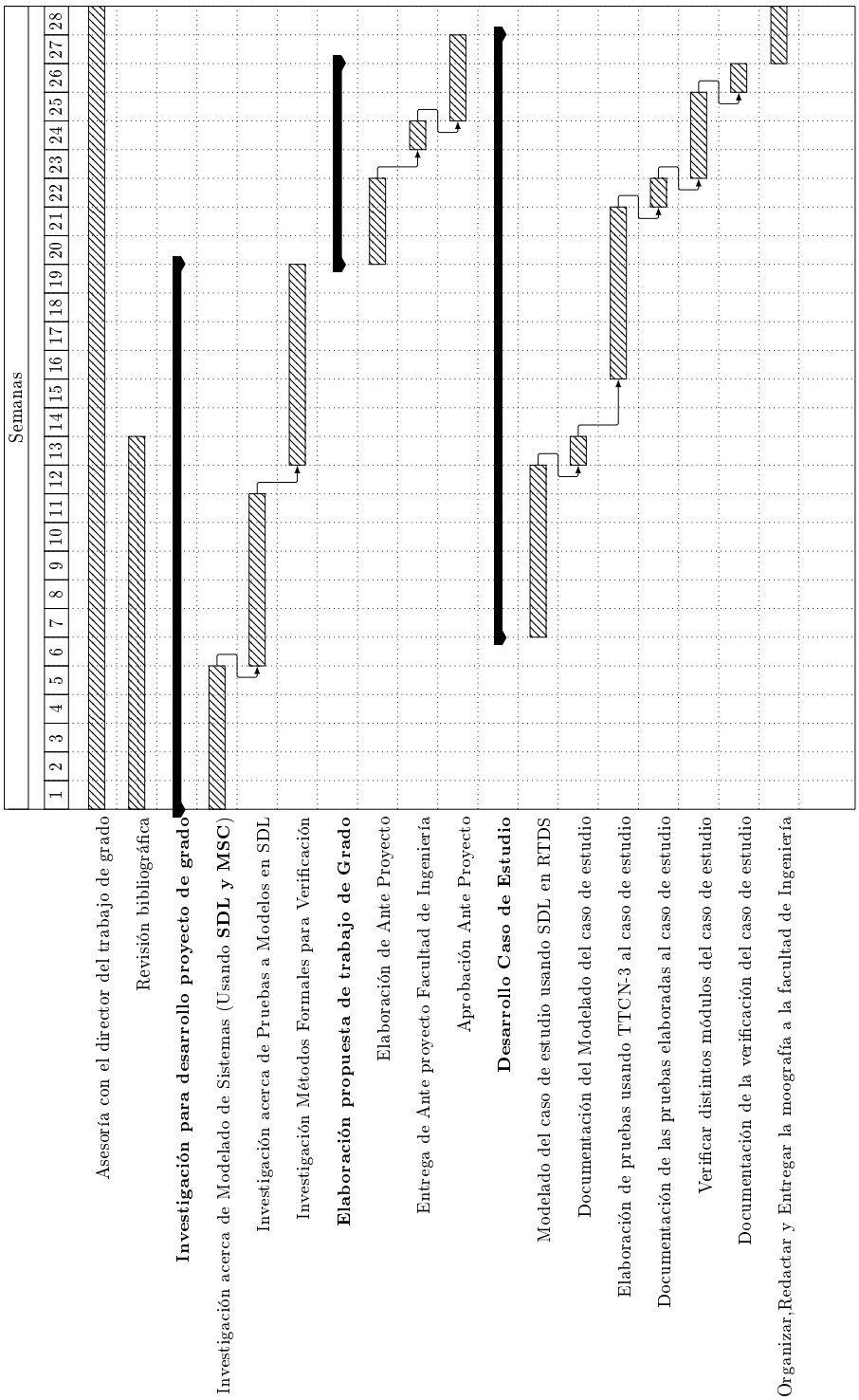

Figura 2.7: Cronograma

Bibliografía

[AO08] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 1 edition, 2008.

[AR11] Lehtmets Andrus and Anna Rannaste. TTCN-3 Basic Introduction. pages 1–17, 2011.

[Ari12] Jaime Arias. Model Checking for tcc Calculus. Technical report, Programa de Ingeniería Electrónica, Pontificia Universidad Javeriana Cali, Cali, 2012.

[BBC+02] J.P. Bowen, K. Bogdanov, J.A. Clark, M. Harman, R.M. Hierons, and P. Krause. FORTEST: formal methods and testing. In *Proceedings 26th Annual International Computer Software and Applications*, pages 91–101. IEEE Comput. Soc, 2002.

[BGI+04] Marius Bozga, Susanne Graf, Ober Ileana, Ober Iulian, and Sifakis Joseph. *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[Bow00] Jonathan P. Bowen. The Ethics of Safety-Critical Systems. *Communications of the ACM*, 43:91—-97, 2000.

[BP06] Adolfo Bravo and Jaime Parra. Verificación formal de Sistemas. Technical report, Programa Computación,Universidad Autonoma Metropolitana Unidad de Iztapalapa, Mexico, 2006.

[CW96] Edmund M. Clarke and Jeannette M. Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, December 1996.

[Ebn04] Michael Ebner. TTCN-3 Test Case Generation from Message Sequence Charts. In *In Workshop on Integrated-reliability with Telecommunications and UML Languages (ISSRE04:WITUL)*, 2004.

[ETS] ETSI's TTCN-3.org Editorial Team. Introduction TTCN-3.

[Fix08] Limor Fix. *25 Years of Model Checking*, volume 5000 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, January 2008.

[GHR+03] Jens Grabowski, Dieter Hogrefe, György Réthy, Ina Schieferdecker, Anthony Wiles, and Colin Willcock. An introduction to the testing and test control notation (TTCN-3). *Computer Networks*, 42(3):375–403, June 2003.

[GJ01] Susanne Graf and Guoping Jia. Verification Experiments on the {Mascara} Protocol. In *Model Checking Software, 8th International SPIN Workshop, Toronto, Canada, May 19-20, 2001, Proceedings*, volume 2057 of *LNCS*. Springer Verlag, May 2001.

[Ham09] Edgardo Hames. Falluto : Un model checker para la verificación de sistemas tolerantes a fallas. Technical report, Facultad de matemática, Astronomía y Física,Universidad Nacional de Córdoba, Cordoba, Argentina, 2009.

[HKL+09] Robert M. Hierons, Paul Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy Vilkomir, Martin R. Woodward, Hussein Zedan, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, and Kalpesh Kapoor. Using formal specifications to support testing. *ACM Computing Surveys*, 41(2):1–76, February 2009.

[Hoa96] C.A.R Hoare. *How Did Software Get So Reliable Without Proof?*, volume 1051 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[Hom13] Bernard Homès. *Fundamentals of Software Testing*. John Wiley & Sons, Inc, 2013.

[IEE94] IEEE Guide for Software Verification and Validation Plans. pages i–87, 1994.

[ISO13] Software and systems engineering Software testing Part 1:Concepts and definitions. pages 1–64, September 2013.

[IT02] ITU-T. *ITU-T Rec. Z.100 – Formal description techniques (FDT) – Specification and Description Language (SDL)*, 2002.

[Kro99] Thomas Kropf. *Introduction to Formal Hardware Verification*. Springer-Verlag New York, Inc., 1st edition, 1999.

[Mar03] Călin Jebelean Marius Minea. Experience with Formal Verification of SDL Protocols. In *Proceedings of the NATO Advanced Research Workshop on Concurrent Information Processing and Computing*, pages pp. 185—-192, Romania, 2003. Al. I. Cuza University Press.

[MMT09] Iván Georgiev Mikovski, José Antonio González Martínez, and Nicolás Mon Trotti. FlexiMC Framework: framework flexible para model checking. 2009.

[PT06] Sergio Pérez and Arturo Terceros. Validación de Modelos Uando IF Toolbox. Technical report, Programa de Ingeniería Electrónica, Universidad Autónoma Metropolitana Unidad Iztapalpa, Mexico, 2006.

[Sch04] Klaus Schneider. *Verification of Reactive Systems: Formal Methods and Algorithms*. Springer, 2004.

[Sch10] Ina Schieferdecker. Test Automation with TTCN-3: State of the Art and a Future Perspective. In *ICTSS 2010, Test Automation with TTCN-3*, Natal, Brazil, 2010.

[Ser12] Alexei Serna. Formal Methods in Industry. *Revista Antioqueña de las Ciencias Computacionales y la Ingeniería de Software RACCIS*, 2(2):44–51, 2012.

[VVBK05] Boštjan Vlaovič, Aleksander Vreže, Zmago Brezoçnik, and Tatjana Kapus. Verification of an SDL Specification – a Case Study. *Electrotechnical Review*, 72:14–21, 2005.

[WDT+11] Colin Willcock, Thomas Deiß, Stephan Tobies, Stefan Keil, Federico Engler, and Stephan Schulz. *An Introduction to TTCN-3*. John Wiley & Sons, second edi edition, 2011.

[WLBF09] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys*, 41(4):1–36, October 2009.