DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN DE PRUEBAS DE COMUNICACIÓN PARA PROTOCOLO MODBUS SERIAL ASCII, SERIAL RTU Y TCP DE LIBRE DISTRIBUCIÓN.

CAROL LISET JAIMES VEGA
ÁNGEL ANDRÉS VARGAS ESTEBAN

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2010

DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN DE PRUEBAS DE COMUNICACIÓN PARA PROTOCOLO MODBUS SERIAL ASCII, SERIAL RTU Y TCP DE LIBRE DISTRIBUCIÓN.

CAROL LISET JAIMES VEGA
ÁNGEL ANDRÉS VARGAS ESTEBAN

Trabajo de grado para optar al título de Ingeniero de Sistemas

Director
Msc. Fernando Antonio Rojas Morales
Maestría en Informática

Codirector
Msc. Pedro Javier Trujillo Tarazona
Maestría en Informática

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2010

A Dios por la vida,

A mi padre Robiel por ser fuente de tantas enseñanzas y hacedor de fortaleza,

A mi madre Dora por ser el pilar de mi vida y absoluto e inmejorable motivo de inspiración,

A toda mi Familia y Amigos por siempre estar ahí...

CAROL LISET

Este trabajo va dedicado a todas las personas que me apoyaron y acompañaron durante este proceso de formación y en particular a:

Mi madre Blanca Nury y a mi padre Ángel Ovidio, ambos mi ejemplo a seguir.

Mis hermanos Diego Alejandro y Diana Carolina con los que siempre he podido contar.

Mi querida Carola, por toda su paciencia y apoyo durante todo mi proceso de formación como profesional.

Para ellos y para todos los que siempre confiaron, GRACIAS!

ÁNGEL ANDRÉS

AGRADECIMIENTOS

A los profesores FERNANDO ANTONIO ROJAS MORALES y PEDRO JAVIER TRUJILLO TARAZONA, por el apoyo, la orientación y la confianza puesta en nosotros.

Al ingeniero JOSÉ NIKOLAI ORTIZ por su tiempo y valiosos aportes a este proyecto, el cual sin su guía y sugerencias no hubiera sido posible.

A SEAL Ltda. por brindarnos la oportunidad desarrollar un proyecto aplicable a ámbito técnico real y por facilitarnos los recursos necesarios para su consecución.

A la UNIVERSIDAD INDUSTRIAL DE SANTANDER, por hacernos profesionales de calidad.

CONTENIDO

INTRODUCCIÓN 1
1. ANTECEDENTES DEL PROYECTO 2
1.1 JUSTIFICACIÓN 2
1.2 OBJETIVOS 3
1.2.1 Objetivo general 3
1.2.2 Objetivos específicos 3
2. MARCO DE REFERENCIA 4
2.1 PROTOCOLO DE COMUNICACIONES 4
2.1.1 Interconexión de Sistemas Abiertos 5
2.2 PROTOCOLO DE COMUNICACIONES MODBUS 7
2.2.1 Funcionamiento del protocolo Modbus 8
2.2.2 Almacenamiento de los datos 10
2.2.3 Id del esclavo 10
2.2.4 Código de función 10
2.2.5 Campo de corrección de errores 11
2.2.5 Modos de comunicación serial 11
2.2.6.1 Modo ASCII 11
2.2.6.2 Modo RTU 12
2.2.7 Estructura del mensaje Modbus en transmisión serial 12
2.2.7.1 Estructura del mensaje en modo ASCII 13
2.2.7.2 Estructura del mensaje RTU 13
2.2.8 Campos del mensaje en transmisión serial 14
2.2.8.1 Campo de direcciones 14
2.2.8.2 Campo de código de función 14
2.2.8.3 Contenido del campo de datos 15
2.2.8.4 Contenido del campo de corrección de errores 15
2.2.9 Transmisión serial de los caracteres 16
2.2.10 Métodos de corrección de errores 16
2.2.10.1 Corrección por paridad 17

2.2.10.2 Verificación de redundancia longitudinal LRC....17
2.2.10.3 Verificación de redundancia cíclica CRC....18
2.2.11 Modo de transmisión TCP/IP....19
2.2.11.1 Modelo cliente – servidor....20
2.2.11.2 Estructura del mensaje Modbus TCP/IP....21
2.2.11.3 El encabezado MBAP....23
2.3 GENERALIDADES SISTEMAS SCADA *(Supervisory Control and Data Acquisition)*....24
2.3.1 HMI *(Human Machine Interface)*....26
2.4 BIBLIOTECA NMODBUS....27
2.5 LICENCIAMIENTO....27
3. METODOLOGÍA....28
3.1 METODOLOGÍA FDD *(FEATURE DRIVEN DEVELOPMENT)*....28
3.2 FUNDAMENTOS DE LA FDD....29
3.2.1 Modelado de objetos del dominio....29
3.2.2 Desarrollo por características....30
3.2.3 Propietario de clases individuales (código)....31
3.2.4 Equipos de características....32
3.2.5 Inspecciones....32
3.2.6 Construcciones regulares programadas....33
3.2.7 Manejo de configuraciones....34
3.2.8 Reporte – visibilidad de los resultados....34
3.3 PROCESOS DE LA FDD....34
3.3.1 Desarrollo de un modelo del objeto general....34
3.3.2 Construcción de la lista de características....35
3.3.3 Planear por característica....35
3.3.3 Diseñar y construir por características....35
4. TRABAJO PRELIMINAR....36
4.1 PREPARACIÓN DE RECURSOS....36
4.1.1 Migración de la librería NModbus 1.8.0 al Framework Mono 2....36
4.1.1.1 Análisis inicial de los ensamblados .Net a través de la utilidad MoMA (Mono Migration Analyzer)....36

4.1.1.2 Apertura inicial de la solución en MonoDevelop 39
4.1.1.3 Selección del Framework y primer intento de compilación 39
4.1.1.4 Corrección de errores de compilación y llamadas a la plataforma (P/Invokes).....40
4.1.1.5 Análisis final de los ensamblados resultado de la compilación 42
5. DESARROLLO DEL PROYECTO.....44
5.1 DESARROLLO DEL MODELO DEL OBJETO GENERAL 44
5.1.1 Plantilla tareas primer proceso de la FDD.....44
5.1.2 Desarrollo tareas plantilla primer proceso de la FDD.....46
5.1.2.1 Formación del equipo de desarrollo.....46
5.1.2.2 Desarrollo de la guía del dominio.....47
5.1.2.3 Documentos de estudio 49
5.1.2.4 Desarrollo del modelo 49
5.1.2.5 Refinar el modelo de objeto general 53
5.1.2.6 Notas escritas del modelo.....53
5.2 CONSTRUCCIÓN DE LA LISTA DE CARACTERÍSTICAS.....55
5.2.1 Plantilla segundo proceso de la FDD 55
5.2.2 Desarrollo tareas plantilla segundo proceso de la FDD 57
5.2.2.1 Formación del equipo de lista de características 57
5.2.2.2 Construcción de la lista de características.....57
5.3 PLANEACIÓN POR CARACTERÍSTICA.....60
5.3.1 Plantilla tercer proceso de la FDD.....60
5.3.2 Desarrollo tareas plantilla tercer proceso de la FDD 62
5.3.2.1 Formación del equipo de planeación 62
5.3.2.2 Secuencia de desarrollo.....62
5.3.2.3 Asignación de los Feature Sets a los programadores jefe.....67
5.3.2.4 Asignación de clases a los desarrolladores 67
5.4 DISEÑO Y CONSTRUCCIÓN POR CARACTERÍSTICA 67
5.4.1 Aspectos relevantes sobre el desarrollo de Simbus.....68
5.4.1.1 Subversion SVN.....68
5.4.1.2 Librerías GTK+ y Gtk# 68
5.4.2 Plantillas cuarto y quinto proceso de la FDD.....69

5.4.2.1 Plantilla cuarto proceso de la FDD....69
5.4.2.2 Plantilla quinto proceso de la FDD....71
5.4.3 Iteración característica seleccionada....72
5.4.3.1 Conducción guía del dominio....72
5.4.3.2 Refinación del modelo de operación....74
5.4.3.3 Asignación de clases a los desarrolladores....75
5.4.3.4 Escritura de encabezados....76
5.4.3.5 Implementar clases y métodos....76
5.4.3.6 Prueba unitaria....80
5.4.3.7 Promoción a construcción....83
6. RESULTADOS FINALES....84
6.1 CARACTERÍSTICAS FINALES SIMBUS....84
6.2 PRUEBAS....98
6.2.1 PLAN DE PRUEBAS....98
7. RECOMENDACIONES....111
8. CONCLUSIONES....112
BIBLIOGRAFÍA....113
ANEXOS....115

LISTA DE TABLAS

Tabla 1. Jerarquías del modelo de referencia OSI. 6
Tabla 2. Tablas de almacenamiento. 10
Tabla 3. Códigos de función protocolo Modbus 11
Tabla 4. Campos del encabezado MBAP 24
Tabla 5. Integrantes equipo de desarrollo 46
Tabla 6. Secuencia de desarrollo 63
Tabla 7. Asignación de clases 75
Tabla 8. Relación característica y resultados 87
Tabla 9. Módulo de pruebas Modbus Master 101
Tabla 10. Módulo de pruebas Modbus Slave 106
Tabla 11. Módulo Integridad 110

LISTA DE FIGURAS

Figura 1. Conector serial para transmisión Modbus 8
Figura 2. Señal utilizada para el envío de datos con el protocolo Modbus. 8
Figura 3. Ciclo de consulta – respuesta protocolo Modbus. 9
Figura 4. Estructura del mensaje en modo ASCII.....13
Figura 5. Estructura del mensaje en modo RTU.....14
Figura 6. Orden de los bits en modo ASCII 16
Figura 7. Orden de los bits modo RTU 16
Figura 8. Secuencia de caracteres LRC 18
Figura 9. Secuencia de caracteres CRC.....19
Figura 10. Arquitectura de comunicación Modbus TCP/IP 21
Figura 11. Unidad de datos Modbus general.....21
Figura 12. Estructura del mensaje Modbus, petición o respuesta sobre TCP/IP ...22
Figura 13. Cambio del mensaje Modbus RTU al mensaje Modbus TCP/IP.....23
Figura 14. Diagrama simplificado de un sistema SCADA.....26
Figura 15. Ciclo de vida FDD según Coad.....29
Figura 16. Paso dos MoMA 36
Figura 17. Paso tres MoMA 37
.Figura 18. Reporte detallado de la herramienta MoMA 38
Figura 19. Selección del Framework y primer intento de compilación 39
Figura 20. Errores de compilación NModbus.sln 40
Figura 21. Corrección línea de código 94 del archivo DataStore.cs 41
Figura 22. Biblioteca FTD2XX.dll.....42
Figura 23. Reporte final MoMA 43
Figura 24. Primer esquema del modelo del dominio.....50
Figura 25. Modelo del dominio primera etapa de modelado.....51
Figura 26. Modelo del dominio o modelo de operación 52
Figura 27. Modelo del operación refinado.....53
Figura 28. Diagrama de secuencia para la característica seleccionada 73
Figura 29. Detalle componente Modcore.....75
Figura 30. Esqueleto métodos interfaz ISlaveDevice 76

Figura 31. Esqueleto clase SlaveWorker 77
Figura 32. Clase SlaveWorker 78
Figura 33. Clase TcpSlave 79
Figura 34. Caso agregado en la clase PanelDevMode 79
Figura 35. Caso agregado en la clase PanelSimulation 80
Figura 36. Simbus GUI, ingreso información de IP y puerto 81
Figura 37. Simbus GUI, dispositivo agregado a la lista de dispositivos 82
Figura 38. Diagrama de clases Simbus 85
Figura 39. Diagrama de clases interfaz gráfica Simbus 86
Figura 40. Árbol de la solución Simbus 93
Figura 42. Vista por defecto de Simbus 96
Figura 43. Vista paneles reubicados y desacoplados 97
Figura 44. Manual de usuario 98
Figura 45. Montaje pruebas simulación serial RTU y ASCII. 99
Figura 46. SGGT Versión 1.0 EP9302A ARM BASED 100
Figura 47. SGGT Versión 1.0 conectada a la red de área local 100

LISTA DE ANEXOS

ANEXO 1: GLOSARIO DE TÉRMINOS.....115
ANEXO 2: DISEÑO PLAN DE PRUEBAS.....119
ANEXO 3. RESUMEN MANUAL DE USUARIO 129
ANEXO 4: DIAGRAMAS DE CLASE.....135
ANEXO 5: DIAGRAMAS DE SECUENCIA.....137

RESUMEN

TÍTULO: DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN DE PRUEBAS DE COMUNICACIÓN PARA PROTOCOLO MODBUS SERIAL ASCII, SERIAL RTU Y TCP DE LIBRE DISTRIBUCIÓN.[*]

AUTORES: CAROL LISET JAIMES VEGA [**]
ANGEL ANDRES VARGAS ESTEBAN[**]

PALABRAS CLAVES: Modbus, Open Source, simulación, FDD, HMI, PLC, RTU.

DESCRIPCIÓN

Este proyecto surge de la necesidad de una herramienta de fácil acceso para la realización de pruebas de comunicación para protocolo Modbus, con un planteamiento basado en las ideologías Open Source o de código abierto que permitan un continuo mejoramiento y difusión de la solución. Simbus se apoya en proyectos Open Source previamente desarrollados que cubren desde el diseño, documentación e implementación. En el diseño se destaca NetBeans como herramienta de modelado y documentación, la implementación se logra gracias al excelente entorno integrado de desarrollo MonoDevelop; a su vez para un control general se hizo uso de el sistema de control de versiones Subversion, con un servidor centralizado provisto por SourceForge el mayor anfitrión y patrocinador de proyectos de código abierto a nivel internacional. Gracias también a la librería NModbus se hizo posible la consecución de los objetivos al implementar gran parte del protocolo de comunicaciones Modbus en el lenguaje de programación C#. Este lenguaje es la base de la solución ya que las librerías Mono permiten que aplicaciones desarrolladas en C# sean compiladas y ejecutadas sobre múltiples plataformas.

El desarrollo de Simbus se enmarca bajo el paradigma del desarrollo metodológico ágil planteado por la FDD Feature Driven Development . Simbus responde a la necesidad de una herramienta de simulación del protocolo Modbus en sus modos de operación Master y Slave, y los tipos de comunicación Serial Ascii, Serial Rtu y Tcp. Simbus permite realizar pruebas de lazos de comunicación y configuraciones en hardware y software que incluyan este protocolo, tales como PLC o instrumentos industriales de medición y paquetes de software HMI.

Por último con el desarrollo de esta herramienta se pretende dar una base para la difusión del protocolo Modbus, usado universalmente en la industria.

---

[*] Trabajo de grado: Modalidad Investigación.

[**] Facultad de Ingenieras Físico Mecánicas, Escuela de Ingeniería de Sistemas e Informáticas. Director: Fernando Antonio Rojas Morales, codirector: Pedro Javier Trujillo Tarazona.

SUMMARY

TITLE: DESIGN AND IMPLEMENTATION OF A FREE DISTRIBUTION COMMUNICATION TEST APPLICATION FOR SERIAL ASCII, SERIAL RTU AND TCP MODBUS PROTOCOL. *

AUTHORS: CAROL LISET JAIMES VEGA **
ANGEL ANDRES VARGAS ESTEBAN**

KEY WORDS: Modbus, Open Source, simulation, FDD, HMI, PLC, RTU.

DESCRIPTION:

This project comes up from the necessity of an easy access tool to perform Modbus protocol communication tests; based in the Open Source ideology the project is open to a constant improvement and diffusion. Simbus is supported by previously developed Open Source projects which are focused in many areas like design, documentation and implementation. For the design, NetBeans has a relevant place as a modeling and documentation tool. The implementation is achieved through the excellent MonoDevelop IDE, and for a general control, the version control system Subversion is used with a central server provided by SourceForge, the biggest international Open Source projects hosting and sponsor. Thanks to the NModbus library the objectives consecution is possible, NModbus implements a big part of the Modbus protocol in the programming language C#. This language is the base of the solution, the Mono libraries allow that C# developed applications be compiled and ran over multiplatform systems.

The Simbus development is guided for the agile methodology FDD Feature Driven Development. Simbus is a Modbus protocol simulation tool, its operation modes are Master and Slave, and the available communication types are Serial ASCII, Serial RTU and TCP. Simbus allows performing communication tests of hardware and software configurations that include this protocol, such as PLC or industrial measurement instruments, and HMI software packages.

The purpose of the development of this tool is to provide the basis for the knowledge of the Modbus protocol, widely used in the industry around the world.

---

* Thesis: Investigation.

** Physic and Mechanical Engineerings College. Systems Engineering and Informatics School. Director: Fernando Antonio Rojas Morales, codirector: Pedro Javier Trujillo Tarazona.

INTRODUCCIÓN

La automatización y el control de procesos son una constante en la industria actualmente, el desarrollo de software para el apoyo de estas tareas es un amplio campo de acción para la Ingeniería de Sistemas. La necesidad de desarrollar múltiples herramientas con diferentes fines crece a medida que tanto la electrónica como las comunicaciones y la informática se entremezclan para formar complejos sistemas que requieren de precisión, rendimiento y calidad.

De las innumerables áreas que están presentes dentro de dichos sistemas, la comunicación entre dispositivos es una de las más importantes, la calidad y confiabilidad de la información son características vitales en el adecuado desempeño y productividad de los modernos sistemas de control. El protocolo Modbus es un estándar utilizado en la industria para la comunicación de dispositivos de control electrónicos, por lo tanto hay un sin número de personas dedicadas a trabajar con dispositivos que se comunican a través de él, sin embargo las herramientas disponibles aun son escasas y tienen altos costos de licenciamiento.

El alto costo es una de las limitantes más importantes tanto para el mercado local como para el área académica, este proyecto busca brindar una aplicación para el desarrollo de pruebas de comunicación basadas en el protocolo Modbus.

Además del uso de técnicas de diseño e implementación de alta calidad, este proyecto está desarrollado bajo los lineamientos del software libre, haciendo de él una herramienta asequible, modificable y mejorable

Dentro de este documento se encuentran las bases teóricas y metodológicas que permitieron la realización de la aplicación, además de la descripción de cada uno de los procesos involucrados en la consecución de los objetivos planteados al inicio de este proyecto.

1. ANTECEDENTES DEL PROYECTO

1.1 JUSTIFICACIÓN

El uso de nuevas tecnologías electrónicas e informáticas en el control de dispositivos ha llevado al aumento en el diseño y desarrollo de aplicaciones que permitan realizar dichas operaciones de control y seguimiento, estas a su vez requieren de formas especializadas de comunicación entre dispositivos y computadoras, aquí aparece el protocolo Modbus, el cual se ha convertido en el estándar de la industria y goza con la mayor aceptación y disponibilidad en la conexión de dispositivos electrónicos industriales.

La implementación de un sistema SCADA o el desarrollo de software especializado para el control de los diferentes tipos de sistemas que admite una red Modbus se ha convertido en una labor de dimensiones épicas para sus desarrolladores, ya que su depuración y prueba dependen en parte del adecuado desempeño que muestren en la transmisión y recepción de datos. Estas tareas son imposibles de llevar a cabo sin contar con terminales RTU (Remote Terminal Unit) o robustos y complejos sistemas HMI (Human Machine Interface) que permitan observar si los dispositivos y la red de comunicación trabajan o no correctamente.

Una inmensa limitación para desarrolladores y estudiantes es no poder acceder a dispositivos tales como RTU’s y a costosos sistemas HMI para realizar las pruebas básicas y convenientes a sus diseños. El objetivo de realizar proyectos y software para dichos dispositivos electrónicos industriales genera la necesidad de una herramienta informática que permita simular un dispositivo de comunicaciones en sus diferentes modos, brindando asistencia al desarrollador, en la prueba o depurado de aplicaciones para los dispositivos y pruebas de comunicación para redes basadas en el protocolo Modbus.

La aplicación que se proyecta en este trabajo no busca en ninguna medida equiparar o competir con sistemas avanzados de adquisición y supervisión de datos ya existentes, su fin es aportar una herramienta sencilla de apoyo para el desarrollador y el estudiante, por esto su naturaleza de libre distribución. Es relevante destacar que mediante la producción de una aplicación como la que se plantea en este proyecto, se acercarían desarrolladores y estudiantes al uso de tecnologías que se encuentran en auge y con altos niveles de aprobación en industrias a nivel nacional e internacional. Este tipo de herramientas busca expandir así el campo de acción de profesionales en sistemas y electrónica mostrando nuevas formas para generar tecnología de calidad y en concordancia con las nuevas exigencias y estándares internacionales.

1.2 OBJETIVOS

1.2.1 Objetivo general

Diseñar e implementar una aplicación multiplataforma de libre distribución, para simular un dispositivo de comunicaciones basado en el protocolo Modbus en sus modos de transmisión, Modbus serial ASCII, serial RTU y TCP.

1.2.2 Objetivos específicos

- Portar la biblioteca NModbus en su versión actual, basada en el .Net Framework 2.0 de Microsoft al Framework multiplataforma Mono versión 2.

- Definir los requerimientos específicos de funcionamiento e interfaz para la aplicación.

- Aplicar la metodología FDD (Feature Driven Development) en todo el ciclo de vida de la aplicación.

- Documentar mediante el Lenguaje de Modelado Unificado UML, un modelo de la aplicación, incluyendo aspectos conceptuales tales como las funciones del mismo y especificar los diferentes métodos y procesos que este realiza.

- Haciendo uso de las clases implementadas en la biblioteca NModbus programar los módulos de comunicación Master y Slave para los modos de transmisión.

- Diseñar e implementar la interfaz gráfica de usuario de acuerdo a los requerimientos a determinar.

- Formalizar la documentación necesaria de la aplicación, la cual incluye manual de usuario, código fuente comentado y diagramas UML requeridos por el modelo.

- Validar el correcto funcionamiento de la aplicación en sus modos de transmisión mediante pruebas de comunicación y consultas Modbus estándar.

2. MARCO DE REFERENCIA

Los sistemas computacionales e informáticos son un área del conocimiento que es utilizada en incontables campos de la ciencia y de la industria. Hoy, no se limitan a sistemas de información administrativo o comercial sino que también aparecen en sistemas electrónicos de control y sistemas industriales, que han dejado de estar aislados para llegar a integrarse con los sistemas de información de las empresas brindando mayores facilidades de manejo y por supuesto mayores controles a los empresarios e ingenieros.

En del desarrollo de este proyecto se deben tener en cuenta varios aspectos dentro del campo de la tecnología y el desarrollo de sistemas. En los siguientes apartados se mostrarán las diferentes bases teóricas necesarias para la consecución de los objetivos que se plantean.

2.1 PROTOCOLO DE COMUNICACIONES

Las comunicaciones de datos comenzaron probablemente mucho antes que existieran registros históricos, en la antigüedad las señales de humo, el sonido de un tambor eran herramientas básicas pero efectivas en la transmisión de información elemental para individuos y tribus. Si se limita la transmisión información mediante señales eléctricas para transmitir información codificada en binario, la comunicación de datos tuvo su origen en 1837, con el invento del telégrafo y el desarrollo de la clave Morse, por Samuel F. B. Morse.

La modernización de estos sistemas eléctricos primitivos ha permitido el desarrollo de complejas redes de comunicación de datos que pueden ser tan simples como dos computadores interconectados, o comprender redes más complejas que pueden abarcar una o más computadoras centrales y cientos o hasta miles de computadoras personales, terminales remotas y estaciones de trabajo. Todos estos elementos de red trabajan en forma conjunta para enviar mensajes, estandarizados mediante una regla común denominada protocolo.

Las redes de comunicaciones de datos se utilizan para interconectar casi cualquier clase de equipos de cómputo digital, llegando así sistemas tales como cajeros automáticos conectados con un computador central, computadores personales con las autopistas de información como Internet, estaciones de trabajo con computadores centrales y dispositivos electrónicos de control tales como PLCs[1] con unidades terminales remotas RTU[2] y estas a su vez conectadas con una o más computadoras centrales.

[1] Controlador Lógico Programable *(Programmable Logic Controller)*
[2] Unidades Terminales Remotas *(Remote Terminal Unit)*

El objetivo principal de una arquitectura de red es proporcionar a los usuarios los medios necesarios para establecer la red y estructurar los flujos de datos que viajarán a través de ella. Una arquitectura en general describe la forma en que se estructura una red e incluye el concepto de niveles o capas dentro de la arquitectura. Cada nivel dentro de la red consiste en protocolos específicos, es decir en reglas de comunicación que desempeñan un conjunto dado de funciones.

Los protocolos son arreglos entre personas o entidades lógicas o físicas. Un protocolo es un conjunto de lineamientos acerca de la formalidad o precedencia, por ejemplo un protocolo diplomático o militar. En este caso un protocolo de red de comunicación de datos es un conjunto de reglas que administra el intercambio ordenado de datos sobre una red.

Los protocolos de comunicación se clasifican en general en síncronos y asíncronos. Por regla los protocolos asíncronos usan formato de datos asíncronos y módems asíncronos, mientras que los protocolos síncronos usan un formato de datos síncronos y módems síncronos.

2.1.1 Interconexión de Sistemas Abiertos

El término de Interconexión de Sistemas Abiertos OSI[3], es el nombre dado al conjunto de normas para las comunicaciones entre computadoras su objetivo principal es contar con un lineamiento estructural para el intercambio de información entre computadoras, terminales o redes. El OSI está patrocinado por ISO[4] y también por la ITU-T[5] , que trabajaron en conjunto para establecer un grupo de de normas OSI y recomendaciones ITU-T.

De esta forma aparece la jerarquía de protocolos OSI, desarrollada para facilitar las comunicaciones, separando las responsabilidades en las redes de computadores en siete capas distintas. El concepto básico de estratificar las responsabilidades es que cada una agregue valores a los servicios suministrados por las capas inferiores. Así el nivel más alto cuenta con todos los servicios necesarios para hacer funcionar una aplicación de datos distribuidos.

En la tabla 1 se muestra el modelo OSI, de interconexión de sistemas abiertos con siete capas. Existen muchas ventajas al hacer uso de la arquitectura estratificada en el modelo OSI. Las diversas capas permiten que se comuniquen diversas computadoras en distintos niveles. Una desventaja de la arquitectura de siete capas es la gran cantidad de encabezados que se le agregan a la información, por ésto si se tiene en cuenta las siete capas, menos del 15% del mensaje transmitido

[3] *Open Systems Interconnection*.
[4] Organización Internacional para la Estandarización *(International Organization for Standarization)*
[5] UIT-T: Sector de Normalización de las Telecomunicaciones (antes CCITT).

es información de la fuente, el resto es indirecto. En la tabla 1 se puede también apreciar el resultado de estos encabezados.

Tabla 1. Jerarquías del modelo de referencia OSI.

Proceso de Aplicaciones									Datos	Proceso de Aplicaciones
Nivel 7 Aplicaciones	Encripción E-E							AH	Datos	Nivel 7 Aplicaciones
Nivel 6 Presentación	Sintaxis, gráficos						PH	AH	Datos	Nivel 6 Presentación
Nivel 5 Sesión	Disponibilidad					SH	PH	AH	Datos	Nivel 5 Sesión
Nivel 4 Transporte	Ruta, recuperación de errores				TH	SH	PH	AH	Datos	Nivel 4 Transporte
Nivel 3 Red	Marcar,línea privada, PDN			NH	TH	SH	PH	AH	Datos	Nivel 3 Red
Nivel 2 Enlace de datos	HDLC		LH	NH	TH	SH	PH	AH	Datos	Nivel 2 Enlace de datos
Nivel 1 Físico	RS-232	PH	LH	NH	TH	SH	PH	AH	Datos	Nivel 7 Físico
Sistema A										Sistema B
AH = encabezado de aplicaciones, PH = encabezado de presentación, SH = encabezado de sesión, TH = encabezado de transporte, NH = encabezado de red, LH = encabezado de enlace, PH = encabezado físico.										

Fuente: *Sistemas de Comunicaciones Electronicas. Wayne Tomasi*.

Una descripción general del modelo es el siguiente.

1. *Capa física:* La capa física es el nivel más bajo en la cual se especifican las normas de los aspectos físicos, eléctricos mecánicos, funcionales y de procedimiento para la transmisión de los datos.

2. *Capa de enlace de datos:* Es la capa responsable de las comunicaciones entre nodos primarios y secundarios de la red. La capa de enlace proporciona un medio para activar, mantener y desactivar el enlace de datos, también proporciona la trama final de la envolvente de información, facilita el flujo ordenado de datos entre nodos y permite la detección y corrección de errores.

3. *Capa de red:* La capa de red determina la configuración que es más adecuada para la función que proporciona la red. A su vez esta capa define la forma como los mensajes se dividen en paquetes de datos, y se conducen de un nodo de transmisión a uno de recepción, dentro de una red de comunicaciones.

4. *Capa de transporte:* Esta capa controla la integridad del mensaje, de principio a fin, la segmentación y la recuperación ante errores para el mensaje.

5. *Capa de sesión:* Es la responsable de la disponibilidad de la red (capacidad de almacenamiento y del procesador). Tiene bajo su responsabilidad los procesos de entrada y salida de la red, y la verificación de usuarios.

6. *Capa de presentación:* Realiza toda conversión de código de sintaxis necesaria para presentar los datos a la red, en un formato común para las comunicaciones. La capa de presentación hace la traducción del conjunto de caracteres, y determina el mecanismo de presentación de los mensajes.

7. *Capa de aplicación:* Es la referente a las aplicaciones y servicios para el usuario.

2.2 PROTOCOLO DE COMUNICACIONES MODBUS

Los controladores programables de Modicon[6] se pueden comunicar entre ellos y con otros dispositivos sobre una gran variedad de redes. Las redes sobre las cuales se pueden comunicar son las redes industriales Modicon Modbus, Modbus Plus y redes estándar como MAP y Ethernet.

Modbus es un protocolo de comunicación desarrollado por sistemas Modicon en 1979, para la gama de controladores lógicos programables de Modicon, ubicado en el nivel siete del Modelo OSI, basado en la arquitectura cliente/servidor. Más adelante fue convertido en un protocolo de comunicaciones estándar en la industria y es el que goza de mayor disponibilidad para la conexión de dispositivos lógicos industriales. Las razones por las cuales Modbus es más usado en comparación con otros protocolos de comunicación son: es público, su implementación es sencilla, requiere poco desarrollo y maneja bloques de datos sin imponer restricciones. Como Modbus es un protocolo abierto, los fabricantes son libres de construir dispositivos que usen este protocolo sin tener que pagar regalías. Esto lo ha convertido en un protocolo muy común, ampliamente usado por muchos fabricantes a través de muchas industrias. Modbus es típicamente usado en transmitir señales desde dispositivos de instrumentación y control de vuelta al controlador principal o sistemas de concurrencia de datos.

En términos simples el protocolo Modbus, es una forma de enviar información entre dispositivos electrónicos. Los puertos estándar en los controladores Modbus usan una interfaz serial compatible RS-232C, figura 1a, la cual define los pines del conector, cableado, nivel de la señal, rata de transmisión en baudios y chequeo de paridad.

El dispositivo que requiere la información en una red utilizando el protocolo Modbus es llamado Modbus Master o Maestro y los dispositivos que entregan la

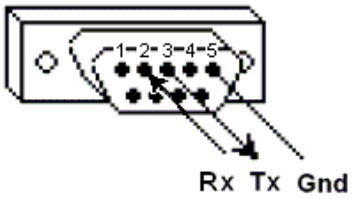
[6] Modicon PLC, primer controlador programable. La marca Modicon ha tenido varios propietarios y hoy en día es propiedad de Scheneider Electric.

información son los Modbus Slaves o Esclavos. En una red Modbus estándar, existe un maestro y hasta 247 esclavos, cada uno con una dirección única de 1 a 247. El maestro también puede escribir información en los esclavos.

2.2.1 Funcionamiento del protocolo Modbus

Modbus transmite sobre líneas seriales entre dispositivos figura 1. La configuración más simple sería un cable serial conectando los dos puertos seriales de dos dispositivos, un Maestro y un Esclavo.

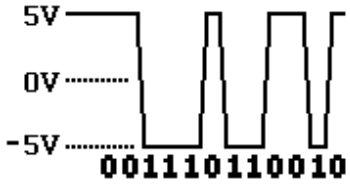
Figura 1. Conector serial para transmisión Modbus



Fuente: *Sitio Web Simply Modbus.*

El dato es enviado como una serie de bits. Cada bit es enviado como un voltaje. Los ceros son enviados como voltajes positivos y los unos como negativos, como se muestra en la figura 2. Los bits son enviados muy rápidamente. La velocidad de transmisión típica es de 9600 bits por segundo.

Figura 2. Señal utilizada para el envío de datos con el protocolo Modbus.



Fuente: *Sitio Web Simply Modbus.*

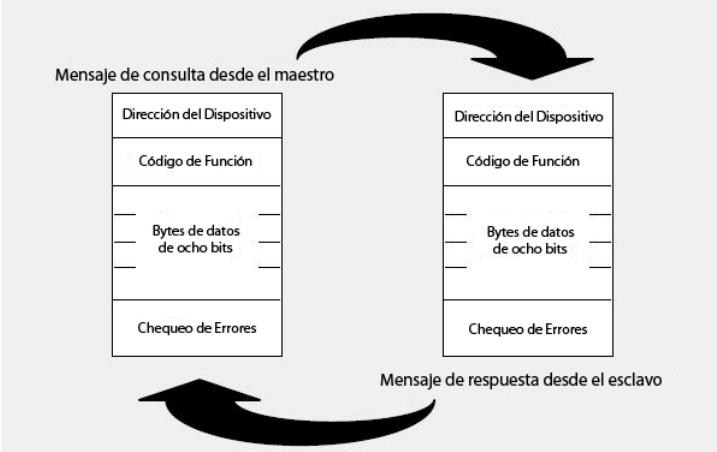
Los dispositivos se comunican usando la técnica cliente/servidor, en la cual solo uno de los dispositivos, el maestro, puede iniciar transacciones llamadas consultas. Los demás dispositivos o esclavos, responden proveyendo los datos requeridos al maestro, o realizando la acción requerida en la consulta.

El maestro puede direccionar individualmente a los esclavos o puede inicializar la emisión de un mensaje a todos los esclavos. Los esclavos devuelven un mensaje llamado respuesta a las consultas que son direccionadas a ellos individualmente.

Cuando la consulta es una emisión del maestro para todos los esclavos, estos no devuelven respuesta.

El protocolo Modbus establece el formato para la consulta del maestro colocando en ésta la dirección del dispositivo, un código de función que define la acción requerida, cualquier dato a ser enviado y un campo de verificación de errores. El mensaje de respuesta del esclavo también es construido usando el protocolo Modbus. Este contiene los campos de confirmación de la acción realizada, cualquier dato a ser devuelto, y un campo de verificación de errores. Si un error ocurre en la recepción del mensaje, o si el esclavo es incapaz de realizar la acción requerida, el esclavo construirá un mensaje de error y enviará este como respuesta. A continuación en la figura 3, se puede observar el ciclo de consulta – respuesta del protocolo Modbus.

Figura 3. Ciclo de consulta – respuesta protocolo Modbus.



Fuente: *Modicon Modbus Protocol Reference Guide.*

En la consulta el código de función dice a la dirección del dispositivo esclavo que tipo de acción realizar. Los bytes de datos contienen cualquier información adicional que el esclavo necesite para realizar la función. Por ejemplo, el código de función 03 pedirá al esclavo que lea determinados registros y que responda con su contenido. El campo de dato debe contener la información que diga al esclavo desde qué registro empezar y cuantos registros leer. El campo de chequeo de error provee un método al esclavo para validar la integridad del mensaje contenido.

Si el esclavo realiza una respuesta normal, el código de función de la misma es un eco del código de función en la consulta. Los bytes de datos contienen los datos recolectados por el esclavo, tales como valores de registros o estados. Si un error

ocurre, la función de código es modificada para indicar que la respuesta es errónea, y los bytes de datos contienen un código que describe el error. El campo de chequeo de error permite al maestro confirmar que el mensaje contenido es válido.

2.2.2 Almacenamiento de los datos

La información es almacenada en el dispositivo esclavo en cuatro diferentes tablas. Dos de estas tablas almacenan valores discretos (coils) y dos almacenan valores numéricos (registers). Las coils y los registers tienen una taba de solo lectura y una tabla de lectura/escritura respectivamente. Cada tabla tiene 9999 valores y cada coil o contact es 1 bit asignado a una dirección entre 0000 y 270E. Cada register es 1 palabra de 16 bits, equivalente a 2 bytes y también tiene una dirección entre 0000 y 270E.

Los números de coil o register pueden ser tomados como números de localización desde que ellos no aparezcan en los mensajes. Por ejemplo, el primer registro de almacenamiento, número 40001, tiene la dirección 0000. La diferencia entre estos dos valores en el offset, determina la tabla a la que pertenece el registro ya que cada tabla tiene un offset diferente 1, 10001, 30001 y 40001.

Tabla 2. Tablas de almacenamiento.

Número de *coil - register*	Dirección de los datos	Tipo	Nombre de la tabla
1 – 9999	0000 a 270E	Lectura – Escritura	Salidas discretas *coils*
10001 - 19999	0000 a 270E	Sólo lectura	Entradas discretas *contacts*
30001 - 39999	0000 a 270E	Sólo lectura	Entradas Análogas *registers*
40001 - 49999	0000 a 270E	Lectura – Escritura	Salidas Análogas *Holding Registers*

Fuente: *Modicon Modbus Protocol Reference Guide.*

2.2.3 Id del esclavo

A cada esclavo en una red se le es asignada una única unidad de dirección desde 1 a 247. Cuando el maestro requiere datos, el primer byte que se envía es la dirección del Esclavo. De esta forma el esclavo sabe después del primer byte si debe o no ignorar el mensaje.

2.2.4 Código de función

El segundo byte enviado por el maestro es el código de función. Este número dice al esclavo a cuál tabla acceder y si necesita leer de ella o escribir en la tabla.

Tabla 3. Códigos de función protocolo Modbus

Código de Función	Acción	Nombre de la tabla
01(01 hex)	Lectura	*Discrete Output Coils* (Salidas Discretas)
05(05 hex)	Escritura simple	*Discrete Output Coil* (Salida Discreta)
15(0F hex)	Escritura múltiple	*Discrete Output Coils* (Salidas Discretas)
02(02 hex)	Lectura	*Discrete Input Contacts* (Entradas Discretas)
04(04 hex)	Lectura	*Analog Input Registers* (Entradas Análogas)
03(03 hex)	Lectura	*Analog Output Holding Registers* (Salidas Análogas)
06(06 hex)	Escritura simple	*Analog Output Holding Register* (Salida Análoga)
16(10 hex)	Escritura múltiple	*Analog Output Holding Register* (Salidas Análogas)

Fuente: *Modicon Modbus Protocol Reference Guide.*

2.2.5 Campo de corrección de errores

El campo de corrección de errores son dos bytes al final del mensaje Modbus, varía su contenido de acuerdo al sistema de control que se esté usando, el cual depende del modo de transmisión que se use, Modbus ASCII o Modbus RTU. Los sistemas utilizados para la corrección de errores son el chequeo de paridad, el método LRC (Longitudinal Redundancy Check), o el método CRC (Cyclical Redundancy Check).

2.2.5 Modos de comunicación serial

Los dispositivos instalados dentro de una red Modbus pueden utilizar dos modos de transmisión serial: ASCII o RTU. Es decisión del usuario determinar qué modo de transmisión utilizar de acuerdo a sus necesidades junto con los parámetros del puerto serial (rata de baudios, modo de paridad, etc.). Es de vital importancia tener en cuenta que tanto el modo de transmisión como los parámetros seriales deben ser los mismos para todos los dispositivos en una red Modbus.

La selección del modo ASCII o RTU pertenece únicamente a las redes Modbus estándar, esta selección define el contenido del los bits de cada campo del mensaje transmitido serialmente en estas redes, también determina como es empaquetada la información dentro de los campos del mensaje y cómo es decodificada.

En otras redes como MAP y Modbus Plus, los mensajes Modbus son colocados dentro de marcos que no están relacionados con la transmisión serial.

2.2.6.1 Modo ASCII

En el caso que los dispositivos sean configurados para comunicarse en una red Modbus usando el modo ASCII (American Standar Code for Information Interchange), cada byte de 8 bits en un mensaje es enviado como dos caracteres

ASCII. La principal ventaja de trabajar en este modo de transmisión es que este permite que ocurran intervalos de tiempo mayores a un segundo entre caracteres sin causar un error. El formato para cada byte en modo ASCII es el siguiente:

Sistema de codificación: Hexadecimal, caracteres ASCII 0 – 9, A – F
Un carácter hexadecimal contenido en cada carácter ASCII del mensaje.

Bits por Byte:
1 bit de inicio
7 bits de datos, el bit menos significante se envía primero.
1 bit para paridad par/impar, ningún bit si no hay paridad.
1 bit de parada, si la paridad es usada, 2 bits si no hay paridad.

Campo de chequeo de errores: Control de redundancia longitudinal. (LRC)

2.2.6.2 Modo RTU

Cuando los dispositivos son configurados para comunicarse en una red Modbus usando el modo Modbus RTU (Remote Terminal Unit), cada byte de 8 bits en un mensaje contiene dos caracteres hexadecimales de 4 bits. La principal ventaja es que al tener una densidad mayor de caracteres, permite una mejor tasa de transferencia que la obtenida en modo ASCII con la misma rata de baudios. Cada mensaje debe ser transmitido en una corriente continua. El formato para cada byte en modo RTU es:

Sistema de codificación: Binario de 8 bits, hexadecimal 0 – 9, A – F
Dos caracteres hexadecimales contenidos en cada campo de 8 bits del mensaje.

Bits por byte:
1 bit de inicio
8 bits de datos, el bit menos significante se envía primero.
1 bit de paridad par/impar, ningún bit si no hay paridad.
1 bit de parada, si la paridad es usada, 2 bits si no hay paridad.

Campo de chequeo de errores: Control de redundancia cíclica (CRC)

2.2.7 Estructura del mensaje Modbus en transmisión serial

Para cualquiera de los dos modos de transmisión serial (ASCII o RTU), un mensaje Modbus es colocado por el dispositivo transmisor en un marco o estructura que tiene un punto inicial y final conocidos. Esto permite que los dispositivos receptores determinen cuál dispositivo está siendo direccionado (o si el mensaje es para todos los dispositivos), también les permite saber cuando el mensaje es completado. Mensajes parciales pueden ser detectados y errores pueden ser tomados como resultado.

2.2.7.1 Estructura del mensaje en modo ASCII

En modo ASCII el mensaje inicia con el carácter columna ASCII (:) (3A hex), y termina con retorno de línea ASCII (CLRF), par (0D y 0A hex), para los demás campos los caracteres permitidos son hexadecimales (0 – 9, A – F). Los dispositivos en la red monitorean continuamente el bus de la red en busca del carácter columna (:), cuando uno es recibido, cada dispositivo decodifica el siguiente campo (campo de dirección) para así saber si esta es la dirección que tiene asignada. En el modo ASCII pueden ocurrir intervalos de un poco más de un segundo entre caracteres sin problema, sin embargo si un intervalo de mayor duración ocurre, el dispositivo receptor asume que ha ocurrido un error. La estructura general del mensaje se puede observar en la figura 4.

Figura 4. Estructura del mensaje en modo ASCII

INICIO	DIRECCIÓN	FUNCIÓN	DATOS	CONTROL LRC	FIN
1 CHAR	2 CHARS	2 CHARS	*N* CHARS	2 CHARS	2 CHARS CRLF

Fuente: *Modicon Modbus Protocol Reference Guide.*

2.2.7.2 Estructura del mensaje RTU

En Modbus RTU, los bytes son enviados consecutivamente sin espacio entre ellos, el carácter de espacio entre los mensajes se utiliza como delimitador, permitiendo al software conocer cuando un nuevo mensaje está iniciando. Cualquier espacio entre bytes causará que Modbus RTU lo interprete como el comienzo de un nuevo mensaje. Esto hace que Modbus RTU trabaje apropiadamente con módems.

Los caracteres permitidos para todos los campos son hexadecimales (0 – 9, A – F), los dispositivos conectados en esta red monitorean continuamente incluyendo los momentos en que hay intervalos de “silencio” o espacios. Cuando el primer campo (campo de direcciones) es recibido, cada dispositivo lo decodifica para encontrar la dirección de determinado dispositivo de la red.

El mensaje completo debe ser transmitido en una corriente continua de bits. Si un intervalo de silencio se presenta y es mayor que 1.5 tiempos de carácter, el dispositivo receptor considerará el siguiente byte como el campo de dirección de un nuevo mensaje. De forma similar si un mensaje empieza antes de que 3.5 tiempos de carácter hayan transcurrido después del mensaje anterior, el dispositivo receptor considerará este una continuación del mensaje anterior, generando un error al final del campo de corrección CRC. La estructura de un mensaje típico se muestra en la figura 5.

Figura 5. Estructura del mensaje en modo RTU

INICIO	DIRECCIÓN	FUNCIÓN	DATOS	CONTROL CRC	FIN
T1-T2-T3-T4	8 BITS	8 BITS	*Nx8 BITS*	16 BITS	T1-T2=T3-T4

Fuente: *Modicon Modbus Protocol Reference Guide.*

2.2.8 Campos del mensaje en transmisión serial

2.2.8.1 Campo de direcciones

Las direcciones en el campo de un mensaje contienen dos caracteres (ASCII) u ocho bits (RTU). Las direcciones válidas para un dispositivo esclavo son asignadas en el rango decimal de 0 a 247. A los dispositivos esclavos individuales les son asignadas las direcciones en el rango de 1 a 247. El maestro direcciona un esclavo colocando la dirección de dicho esclavo en el campo de direcciones del mensaje. Cuando el esclavo envía su respuesta, este coloca su propia dirección en el campo de direcciones de la respuesta para permitir al maestro conocer cual esclavo está dando respuesta. La dirección 0 es usada para enviar el mensaje a todos los dispositivos esclavos ya que está dirección es reconocida por todos.

2.2.8.2 Campo de código de función

El campo del código de función de un mensaje consiste en dos caracteres (ASCII) u ocho bits (RTU). Los códigos válidos están en el rango decimal de 1 a 255. Cuando un mensaje es enviado desde el maestro a un dispositivo esclavo, el campo de código de función le dice al esclavo que tipo de acción llevar a cabo. Cuando el esclavo responde al maestro, este usa el campo de código de función para indicar una respuesta normal o si algún tipo de error ocurre. En una respuesta normal el esclavo simplemente coloca el mismo código de función enviado por el maestro o, en caso de un error, el esclavo devuelve un código equivalente al código de función original pero con su bit más significativo en 1 lógico. Para ver más claramente esta situación se presenta a continuación un ejemplo:

Un mensaje desde el maestro a un esclavo para la lectura de un grupo de registros de almacenamiento tendría el siguiente código de función 0000 0011 (Hexadecimal 03), si el dispositivo esclavo realiza la acción requerida sin error, este retorna el mismo código en su respuesta, de lo contrario retornará el código 1000 0011 (Hexadecimal 83). Adicionalmente a este cambio en el código de función en caso de un error, el esclavo coloca un código único en el campo de datos del mensaje de respuesta. Esto dice al maestro que tipo de error ocurre, o la razón de la excepción.

2.2.8.3 Contenido del campo de datos

El campo de datos es construido usando grupos de dos dígitos hexadecimales, en el rango de 00 a FF. Estos pueden ser hechos a partir de un par de caracteres ASCII, o de un carácter RTU, de acuerdo al modo de transmisión serial usado en la red. Las direcciones del campo de datos enviadas desde el maestro hacia los dispositivos esclavos contienen información adicional la cual el esclavo necesita para realizar la acción definida por el código de función. Esta información puede ser direcciones discretas de registros, la cantidad de registros a ser procesados y la cantidad actual de bytes de datos en el campo.

Por ejemplo, si el maestro requiere que un esclavo lea un grupo de registros de almacenamiento (código de función 03), el campo de datos especifica el registro de inicio y cuantos registros deben ser leídos. Si el maestro escribe en un grupo de registros en el esclavo (código de función 10), el campo de datos especifica el registro de inicio, cuantos registros van a ser escritos, el contador de bytes de datos a seguir en el campo de datos, y el dato a ser escrito en los registros.

Si no ocurre ningún error, el campo de datos en la respuesta del esclavo al maestro contiene el dato requerido, pero en caso de que se presente un error el campo de datos contiene el código de excepción que puede ser usada por el maestro para determinar la siguiente acción que debe realizar.

También es posible que el campo de datos no exista o sea de longitud cero en ciertos tipos de mensajes.

2.2.8.4 Contenido del campo de corrección de errores

Dos tipos de métodos de errores son usadas para las redes estándar Modbus. El campo de corrección de errores depende del método de transmisión que está siendo usado.

Cuando es usado el modo ASCII para estructurar los mensajes, el campo de corrección de errores contiene dos caracteres ASCII. Los caracteres del campo de errores resultan del cálculo de corrección de redundancia longitudinal (LRC) que es desarrollado sobre el contenido del mensaje. Los caracteres LRC son añadidos al mensaje como el último campo que precede los caracteres CRLF.

En el caso de que el modo RTU sea usado para estructurar el mensaje, el campo de corrección de errores contiene un valor de 16 bits, implementado como dos bytes de 8 bits. El valor de corrección de errores es el resultado de un cálculo de corrección de redundancia cíclica (LRC) sobre el contenido del mensaje. El campo CRC es añadido al mensaje como el último campo en el mensaje.

2.2.9 Transmisión serial de los caracteres

Cuando los mensajes son transmitidos sobre una red Modbus serial estándar, cada byte o carácter es enviado en el orden de izquierda a derecha iniciando con el bit menos significativo LSB (Least Significant Bit) y terminando con el bit más significativo MSB (More Significant Bit).

Usando modo ASCII, la secuencia de bits es la siguiente:

Figura 6. Orden de los bits en modo ASCII

Con chequeo de paridad

	Start	1	2	3	4	5	6	7	Par	Stop	

Sin chequeo de paridad

	Start	1	2	3	4	5	6	7	Stop	Stop	

Fuente: *Modicon Modbus Protocol Reference Guide.*

En el caso de RTU, la secuencia de bits es:

Figura 7. Orden de los bits modo RTU

Con chequeo de paridad

	Start	1	2	3	4	5	6	7	8	Par	Stop	

Sin chequeo de paridad

	Start	1	2	3	4	5	6	7	8	Stop	Stop	

Fuente: *Modicon Modbus Protocol Reference Guide.*

2.2.10 Métodos de corrección de errores

Las redes seriales estándar usan dos tipos de corrección de errores. El chequeo de paridad (par o impar) puede ser aplicado de manera opcional a cada carácter. La corrección de la estructura (LRC o CRC) es aplicada al mensaje entero. Ambos tipos de corrección son generados en el dispositivo maestro y aplicados al contenido del mensaje antes de la transmisión. El dispositivo esclavo verifica cada carácter y el mensaje completo durante la recepción.

El maestro es configurado por el usuario para esperar por un intervalo de tiempo predeterminado antes de abortar la transacción. Este intervalo está configurado para ser lo suficientemente largo para que cualquier esclavo responda normalmente. Si el esclavo detecta un error en la transmisión, el mensaje no será ejecutado. El esclavo no construirá una respuesta al maestro, por lo tanto el

tiempo de espera expirará y permitirá al programa del maestro manejar el error. Es importante aclarar que un mensaje que es dirigido a un dispositivo esclavo no existente, también producirá un tiempo de espera.

2.2.10.1 Corrección por paridad

Para visualizar mejor la situación, se tiene el siguiente ejemplo, los bits a continuación están contenidos en un marco de caracteres RTU:

1100 0101

La cantidad total de bits con valor lógico 1 en el marco es de cuatro. Si la paridad par es usada el bit de paridad del marco será un 0. Completando así un total par de bits con valor lógico 1, en este caso 4 bits. Si se usa la paridad impar, el bit de paridad será 1, obteniendo una cantidad impar de cinco bits con valor lógico 1.

Los controladores utilizados en la red Modbus, se pueden configurar con corrección de paridad par o impar o sin corrección de paridad. Esta decisión determinará como el bit de paridad será establecido en cada carácter. Cuando un mensaje es transmitido, el bit de paridad es calculado y aplicado a la transmisión de los bits originales. El dispositivo receptor verifica la cantidad de bits de a cuerdo al tipo de paridad predeterminado y coloca un error si estos no concuerdan con la configuración dada al dispositivo. Dentro de una red Modbus todos los elementos interconectados deben estar configurados para usar el mismo método de corrección de paridad.

Hay que notar que la corrección por paridad solo puede detectar un error si el número de bits perdidos en la transmisión de datos es impar, de lo contrario el error pasará inadvertido, ya que, si en una transmisión con paridad impar y que contiene 3 bits con valor lógico 1 se pierden 2 de estos bits, el conteo de bits seguirá siendo impar para ese único bit 1 que se mantuvo en la transmisión.

Si no existe especificación del tipo de corrección de paridad, no se transmitirá ningún bit de paridad y no se podrá realizar dicha verificación, en su lugar un bit de parada adicional es transmitido para completar el marco del carácter.

2.2.10.2 Verificación de redundancia longitudinal LRC

Los mensajes transmitidos a través de una red configurada en modo ASCII incluyen un campo de error que está basado en el método de verificación de redundancia longitudinal LRC de sus siglas en ingles Longitudinal Redundancy Check. El campo LRC revisa el contenido del mensaje excluyendo el carácter columna inicial y el par CLRF final. La corrección es aplicada sin importar que tipo de paridad si par o impar, se esté aplicando a los caracteres que forman el mensaje.

El campo LRC es un byte que contiene un valor binario de 8 bits. El valor LRC es calculado y añadido al mensaje por el dispositivo transmisor de manera análoga el dispositivo receptor calcula un LRC durante la recepción del mensaje y lo compara con el que está incluido previamente en el mensaje, si los dos valores no coinciden la respuesta será un error.

El cálculo del LRC se realiza sobre el campo de mensaje por medio de la suma sucesiva de los bytes del mensaje y complementando a dos la respuesta. Como el LRC es un campo de 8 bits, cada nueva adición que resulte en un valor mayor al decimal 255 simplemente coloca el valor del campo en cero, finalmente el acarreo de la suma es descartado ya que no existe un noveno bit para contenerlo.

Cuando el campo LRC de 8 bits es transmitido en el mensaje, el caracter de orden más alto será transmitido primero, seguido por los caracteres de más bajo orden. Por ejemplo si el valor LRC es 61 hex (0110 0001) la posición es:

Figura 8. Secuencia de caracteres LRC

:	Dir.	Func.	Contador de Datos	Datos	Datos	Datos	Datos	LRC Hi	LRC Lo	CR	LF
								6	1		

Fuente: *Modicon Modbus Protocol Reference Guide.*

2.2.10.3 Verificación de redundancia cíclica CRC

En modo RTU, el campo de corrección de errores incluido en el mensaje está basado en el método de verificación de redundancia cíclica CRC de sus siglas en inglés Cyclical Redundancy Check. De igual forma el campo de verificación CRC verifica el contenido del mensaje y es aplicado sin importar el método de paridad usado para los caracteres del mensaje. El campo de dos bytes contiene un valor binario de 16 bits, este es calculado por el dispositivo transmisor el cual añade el CRC al mensaje. El dispositivo receptor calcula nuevamente un CRC durante la recepción del mensaje y lo compara con el valor del CRC que es recibido, si los dos valores no son iguales la respuesta será un error.

El campo CRC es inicialmente cargado con todos sus bits en 1, un proceso inicia la aplicación sucesiva de los bytes del mensaje al contenido actual del campo CRC, únicamente los 8 bits de cada carácter son usados para generar el CRC, los bits de inicio, parada y paridad son obviados. Para generar el CRC, cada carácter de 8 bits es operado por una OR exclusiva con el contenido del registro, entonces el resultado es corrido en dirección del bit menos significativo (LSB), con un cero llenando la posición del bit más significativo (MSB). El LSB es extraído y examinado, si el LSB es un 1, al registro se le aplica entonces una OR exclusiva

con un valor fijo programado, si el LSB es un 0, no se realiza ninguna operación OR exclusiva.

Para entender de forma más clara como se genera el CRC, este proceso se describe en los pasos a continuación:

1. Carga del registro CRC de 16 bits con FFFF hex. (todos los bits en 1).

2. Operación OR exclusiva para el primer byte del mensaje con el byte de menor orden del campo CRC de 16 bits, el resultado de esta operación es colocado en el campo CRC.

3. Corrimiento de un registro CRC un bit a la derecha (hacia el LSB) con un cero llenando el MSB, se extrae y se examina el LSB.

4. Si el LSB es 0: Se repite el paso 3 (otro corrimiento).

5. Si el LSB es 1: OR exclusiva del registro con el valor predeterminado A001 (1010 0000 0000 0001).

6. Repetir el paso 3 y 4 hasta completar los 8 corrimientos. Cuando esto se termine, un byte habrá sido procesado.

7. Repita los pasos 2 a 5 para el próximo byte del mensaje y continúe esto hasta que todos los bytes hayan sido procesados.

8. El contenido final del registro CRC es el valor CRC.

9. Cuando el CRC es colocado en el nuevo mensaje, sus bytes bajo y alto deben ser colocados como se describe a continuación.

Para la transmisión del CRC este es ubicado en el mensaje, el byte orden más bajo será transmitido primero, seguido por el bit de orden más alto por ejemplo el valor CRC 1241 hex. (0001 0010 0100 0001) es transmitido así:

Figura 9. Secuencia de caracteres CRC

Dir.	Func.	Cont. Datos	Datos	Datos	Datos	Datos	CRC Lo	CRC HI
							41	42

Fuente: *Modicon Modbus Protocol Reference Guide.*

2.2.11 Modo de transmisión TCP/IP

El Protocolo de Control de Transmisión TCP de sus siglas en ingles Transmission Control Protocol y el Protocolo de Internet IP Internet Protocol son usados juntos y son el protocolo mundialmente adoptado para la red Internet. Cuando la

información de una red Modbus es enviada usando estos protocolos, los datos son pasados a TCP, donde información adicional le es agregada y dada al IP, el cual la coloca dentro de un paquete (o datagrama) y la transmite. El protocolo TCP debe establecer la conexión antes de transferir los datos, siempre y cuando esté basada en el protocolo. El maestro (Cliente en Modbus TCP) establece una conexión con el esclavo (Servidor en Modbus TCP), el Servidor responde a las consultas del Cliente hasta el momento en el que el Cliente cierre dicha conexión.

2.2.11.1 Modelo cliente – servidor

El servicio de mensajes Modbus provee una comunicación Cliente/Servidor entre los dispositivos conectados sobre una red Ethernet TCP/IP. Este modelo cliente servidor está basado en 4 tipos de mensajes:

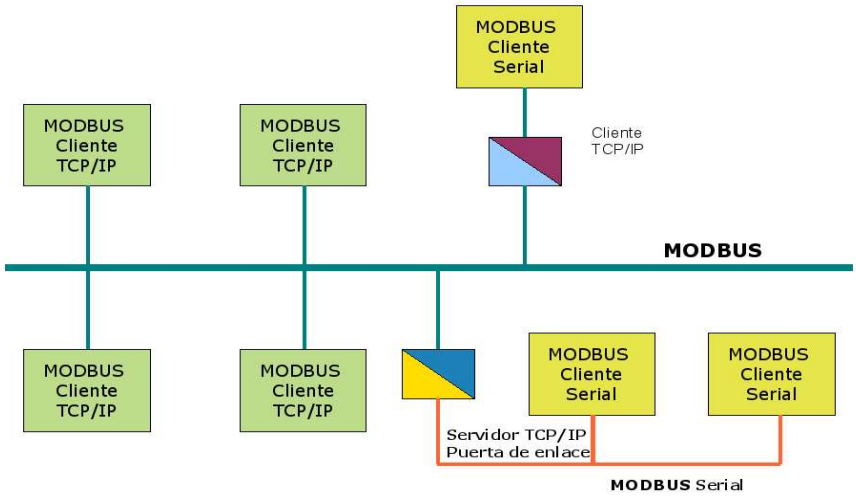
Petición Modbus: es el mensaje enviado sobre la red por el cliente para inicializar una transacción.
Indicación Modbus: es el mensaje de petición recibido en el lado del servidor.
Respuesta Modbus: es el mensaje de respuesta enviado por el servidor.
Confirmación Modbus: es el mensaje de respuesta recibido en el lado del cliente.

Los servicios de mensajes Modbus (Modelo Cliente/Servidor) son usados para el cambio de información en tiempo real entre:

- Dos aplicaciones de dispositivos.
- Aplicación de dispositivo y otro dispositivo.
- Aplicaciones HMI/SCADA y dispositivos.
- Un PC y un programa de dispositivo

Un sistema de comunicación basado en Modbus TCP/IP puede incluir diferentes tipos de dispositivos, entre ellos dispositivos TCP/IP cliente y servidor conectados a una red TCP/IP y dispositivos de interconexión como puertas de enlace, puentes o enrutadores para la conexión entre la red TCP/IP y una línea de subred lo cual permite conexiones con los Clientes Modbus de la línea serial y los dispositivos servidores.

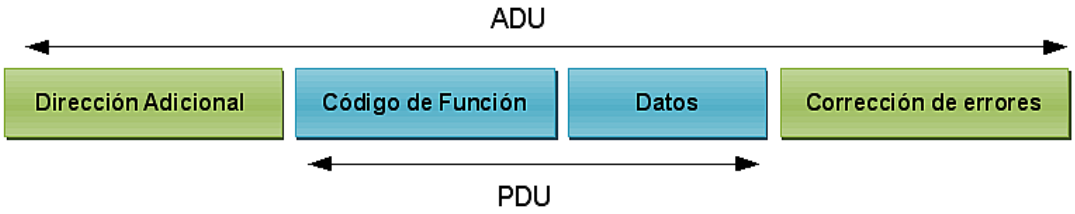
Figura 10. Arquitectura de comunicación Modbus TCP/IP



Fuente: *MODBUS IDA. Modbus Application Protocol Specification*

El protocolo Modbus define una Unidad de Datos del Protocolo simple o PDU (Protocol Data Unit), esta unidad es independiente de las capas de comunicación subyacentes. El trazado del protocolo Modbus sobre buses específicos o redes puede introducir algunos campos adicionales en la Unidad de Datos de la Aplicación ADU (Application Data Unit).

Figura 11. Unidad de datos Modbus general



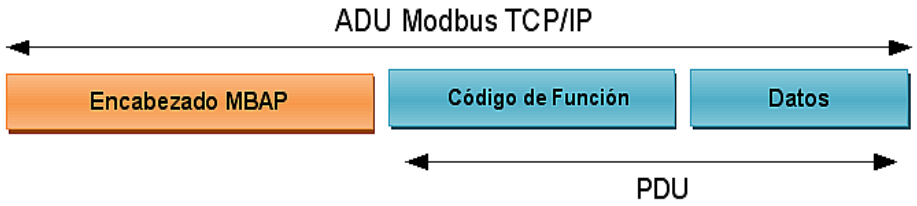
Fuente: *MODBUS IDA. Modbus Application Protocol Specification*

El cliente que inicializa una transacción Modbus construye la Unidad de Datos de Aplicación ADU. El código de función indica al servidor cual tipo de acción realizar.

2.2.11.2 Estructura del mensaje Modbus TCP/IP

La estructura de una petición o respuesta Modbus que es llevada sobre una red TCP/IP tiene algunas características especiales.

Figura 12. Estructura del mensaje Modbus, petición o respuesta sobre TCP/IP



Fuente: *MODBUS IDA. Modbus Application Protocol Specification*

Un encabezado dedicado es usado sobre TCP/IP para identificar la unidad de datos ADU, este encabezado es llamado Cabecera de Protocolo Modbus MBAP (Modbus Application Protocol Header) y tiene algunas diferencias respecto a la ADU usada en Modo RTU, estas son:

- El campo que usualmente en el modo Modbus serial contiene la dirección del esclavo, es reemplazado por un byte individual que contiene el Unit Identifier o Identificador de Unidad dentro del encabezado MBAP. El Identificador de Unidad es usado para comunicar dispositivos como puentes, enrutadores y pasarelas que utilizan una única dirección IP para soportar múltiples unidades Modbus independientes.
- Todas las peticiones y respuestas Modbus están diseñadas de tal forma que el receptor puede verificar que un mensaje ha terminado. Para códigos de función donde la estructura del mensaje Modbus tiene una longitud fija, el código de función es suficiente. Para códigos de función que llevan una cantidad variable de datos en la petición o la respuesta, el campo de datos incluye un conteo de bytes.
- Cuando Modbus está sobre TCP, información adicional es llevada en el encabezado MBAP que permite al receptor reconocer los límites del mensaje, incluso, si este ha sido dividido en múltiples paquetes para su transmisión. La integridad de la transmisión se garantiza gracias a la existencia de reglas explícitas e implícitas sobre la longitud del mensaje y el uso de un código de corrección de errores CRC-32 (en Ethernet), el resultado es una muy baja probabilidad de algún error no detectado en el mensaje de petición o respuesta.

Para comprender mejor, se tiene inicialmente un mensaje Modbus RTU al cual se le remueven la ID del Esclavo, ubicada al comienzo del mensaje y el campo CRC ubicado al final de este, así se obtiene la Unidad de Datos del Protocolo PDU. Por ejemplo en una petición para el contenido análogo de los registros de contención número 40108 hasta 40110 en el Esclavo con dirección 17, el mensaje en Modbus RTU es:

11 03 006B 0003 7687

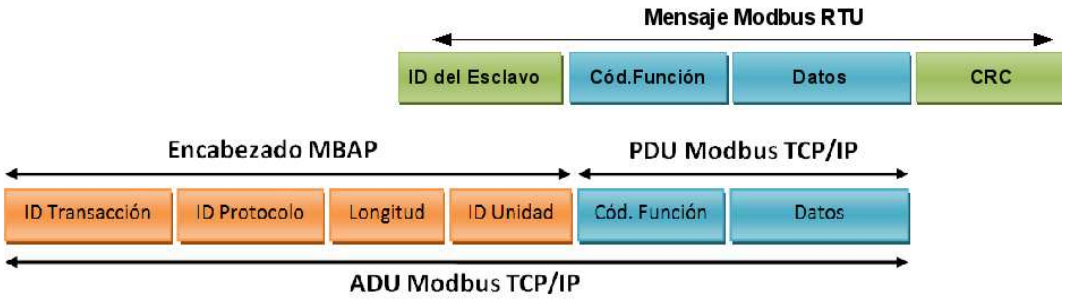
11: Dirección del Esclavo (17 = 11 hex)
03: Código de función (Lectura salida análoga registros contenedores)
006B: Dirección del primer registro requerido (40108–40001 = 107 = 6B hex)
0003: Número total de registros requeridos (Leer 3 registros 40108 a 40110)
7687: Campo CRC

Removiendo el ID del Esclavo y el campo CRC se obtiene:

03 006B 0003

Al resultado que se obtiene se le coloca el encabezado MBAP como se ve en la figura 13.

Figura 13. Cambio del mensaje Modbus RTU al mensaje Modbus TCP/IP



Fuente: *MODBUS IDA. Modbus Application Protocol Specification*

2.2.11.3 El encabezado MBAP

Los campos que componen el encabezado MBAP están cuidadosamente destinados para realizar determinadas funciones, estos son, el Identificador de Transacción, dos bytes enviados por el Cliente para identificar de forma única cada petición, estos bytes son repetidos por el servidor ya que sus respuestas no pueden ser recibidas en el mismo orden que las peticiones, el Identificador de Protocolo, dos bytes asignados por el cliente, en caso del protocolo Modbus TCP su valor es 00 00 hex, el campo de Longitud, también dos bytes que identifican el número de bytes que tendrá el mensaje a seguir incluyendo el Identificador de Unidad y los campos de datos, finalmente, el Identificador de Unidad, un byte asignado por el Cliente y repetido por el Servidor para la identificación de un esclavo remoto conectado sobre línea seria o sobre otros buses, de forma resumida los elementos del encabezado MBAP se muestran en la tabla 4.

Tabla 4. Campos del encabezado MBAP

Campos	Longitud	Descripción	Cliente	Servidor
Identificador de transacción	2 bytes	Identificación de una petición o respuesta Modbus.	Iniciado por el cliente.	Copiado por el servidor desde la petición recibida.
Identificador de protocolo	2 bytes	0 = Protocolo Modbus	Iniciado por el cliente.	Copiado por el servidos desde la petición recibida.
Longitud	2 bytes	Numero siguiente de bytes.	Iniciado por el cliente. (Petición)	Iniciado por el servidor (Respuesta)
Identificador de la unidad	1 byte	Identificación de un esclavo remoto conectado a una línea seria o a otros buses.	Iniciado por el cliente.	Copiado por el servidor desde la petición recibida.

Fuente: *MODBUS IDA. Modbus Application Protocol Specification*

El encabezado MBAP tiene una longitud de 7 bytes y después de ser añadido a la Unidad de Datos del Protocolo (PDU), se obtiene la Unidad de Datos de Aplicación (ADU). Figura 12.

Retomando el ejemplo que se presentó en la sección anterior, con el mensaje Modbus RTU 11 03 006B 0003 7687, el equivalente para este mensaje en Modbus TCP es:

0001 0000 0006 11 03 006B 0003

0001: Identificador de la transacción.
0000: Identificador de protocolo.
0006: Longitud del mensaje (6 siguiente bytes)
11: Identificador de Unidad.
03: Código de función
006B: Dirección del dato del primer registro requerido. (40108-40001 = 107=6B hex)
0003: Número total de registros requeridos (Leer registros 40108 a 40110)

2.3 GENERALIDADES SISTEMAS SCADA *(Supervisory Control and Data Acquisition)*

SCADA (Supervisory Control And Data Adquisition) o en español registro de datos y control de supervisión es principalmente un sistema que monitoriza y controla un proceso ya sea industrial, de infraestructura y de Instalaciones. De forma más simple puede verse como un computador controlando y monitorizando un

proceso. Este proceso puede ser industrial, de infraestructura o para diferentes tipos de complejos habitacionales o de negocios.

SCADA principalmente es un concepto que agrupa o se compone usualmente de los siguientes elementos:

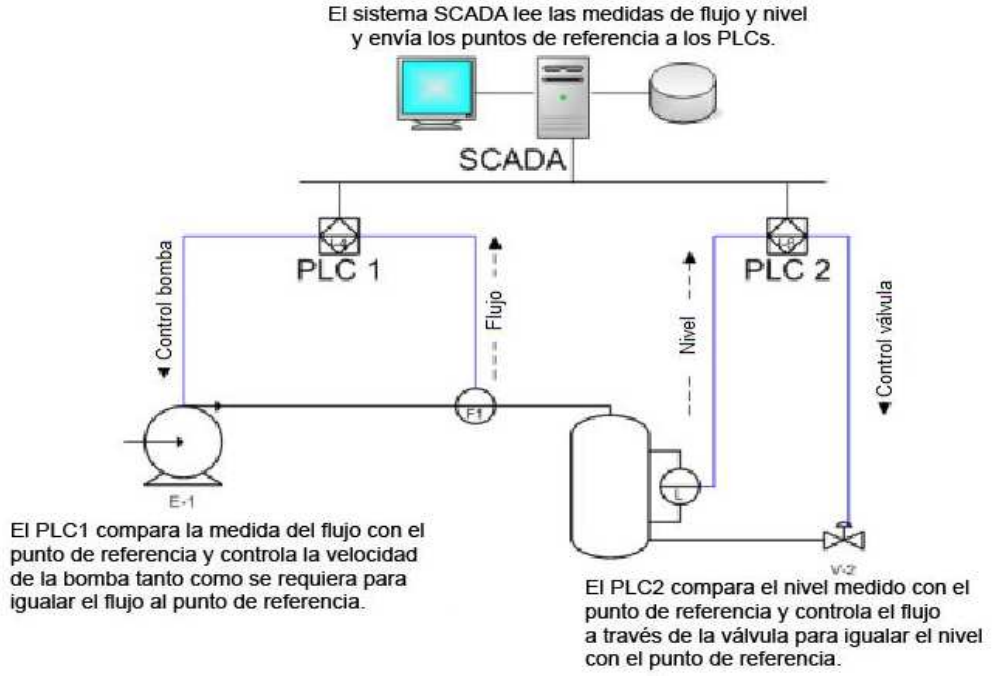
- Software HMI: aparato o software que presenta los datos procesados al operador (humano), y que le permite al mismo controlar el proceso.
- Sistema supervisor: usualmente un computador que obtiene (adquiere) los datos y envía comandos de control al proceso.
- Remote Terminal Unit (RTU): dispositivos a los cuales se conectan los sensores, computadores de flujo y demás dispositivos electromecánicos, convirtiendo sus señales en datos o consultándolos directamente y transmitiéndolos al sistema supervisor.
- Comunicación: la infraestructura que conecta el sistema supervisor, RTU's y demás dispositivos.

El término SCADA usualmente se refiere a sistemas centralizados los cuales monitorizan y controlan sitios enteros o complejos industriales. La mayoría de funciones de control son realizadas automáticamente por las Remote Terminal Unit (RTUs) o por los Programmable Logic Controllers (PLCs). Las funciones de control del servidor están casi siempre restringidas a reajustes básicos del sitio o capacidades de nivel de supervisión. Por ejemplo un PLC puede controlar el flujo de agua fría a través de un proceso, pero también puede permitirle a un operador cambiar el punto de referencia de control para el flujo, el sistema SCADA permitirá grabar y mostrar cualquier condición de alarma como la pérdida de un flujo o una alta temperatura.

La realimentación del lazo de control es cerrada a través del RTU o el PLC, el sistema SCADA monitoriza el desempeño general de dicho lazo muestra gráficas con históricos, tablas, alarmas, eventos, permisos y accesos de los usuarios.

A continuación se presenta un diagrama usual de un sistema SCADA:

Figura 14. Diagrama simplificado de un sistema SCADA



Fuente: *Sitio web Infotech. http://infotechasia.com/scada.htm*

2.3.1 HMI *(Human Machine Interface)*

La interfaz hombre-máquina o HMI es el aparato o software que presenta los datos procesados al operador (humano), y que le permite al mismo controlar el proceso. El termino HMI es usado principalmente para referirse a interfaz de usuarios (User Interface) de sistemas mecánicos o electromecánicos.

Un software HMI esta usualmente enlazado en un sistema SCADA al software del sistema supervisor. Presenta la información de manera gráfica en forma de esquemas que representan el sistema que se está controlando. Por ejemplo, un esquemático que representa una planta y cada uno de los instrumentos que se están supervisando, y sobre cada instrumento se informa del valor actual de presión, temperatura, flujo, rpm's y demás atributos necesarios respectivamente, adicionalmente se muestran alarmas si se presenta un estado anormal en alguno de los instrumentos. Desde el software HMI entonces se podría enviar una orden para cerrar una válvula o cambiar puntos límites en el mismo. Por último un software HMI presenta históricos del proceso almacenados a una determinada frecuencia.

Existen 2 Tipos de software HMI, los desarrollados a la medida y los paquetes HMI que contienen la mayoría de las funciones de los sistemas SCADA que son completamente parametrizables entre los que se encuentran NILookut (National Instruments), iFIX (GE Fanuc), SIMATIC WinCC (SIEMENS) y DeltaV Operator Interface (Emerson).

2.4 BIBLIOTECA NMODBUS

NModbus es una biblioteca de libre distribución que implementa el protocolo modbus en el lenguaje C# 3.0. Soporta los protocolos serial ASCII, serial RTU, serial sobre USB ASCII, serial sobre USB RTU, TCP y UDP. Su uso es bastante sencillo y dadas las pocas llamadas al sistema que realiza se puede pensar en portar la misma a mono 2.4 para lograr obtener una aplicación multiplataforma.

2.5 LICENCIAMIENTO

La Licencia Pública General GNU, por sus siglas en inglés General Public License, o simplemente su acrónimo en inglés GNU GPL es un tipo de licencia creada en la mitad de la década de los 80 por la Free Software Foundation. Su fin es el de proteger la distribución, modificación y uso de software. Tiene como propósito declarar que el software cubierto bajo ella es software libre y protegerlo así de cualquier tipo de apropiación que resulte en la restricción de dichas libertades para los usuarios.

La distribución de esta aplicación se llevará a cabo bajo los lineamientos de la Licencia Pública General GNU GPLv2, en concordancia con la biblioteca NModbus la cual se encuentra distribuida bajo la misma licencia.

3. METODOLOGÍA

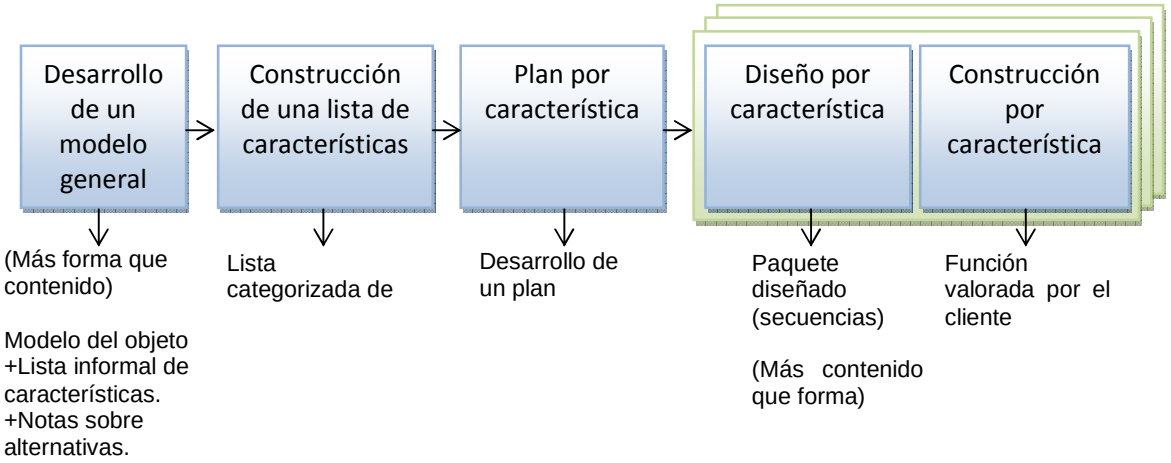
Para el desarrollo de la aplicación planteada en este proyecto, es necesario utilizar una metodología que se ajuste a altos estándares de calidad y que a su vez permita una implementación ágil de la aplicación. Dentro de la teoría de Ingeniería del Software se presentan una gran variedad de las llamadas metodologías ágiles, de las cuales FDD (Feature Driven Development) o Desarrollo Basado en Características es la que se ajusta de mejor forma a las necesidades y características de esta aplicación. A principio de la década de los 90 las metodologías ágiles en el desarrollo de software surgieron en medio de vastas críticas, este nuevo enfoque revolucionó en aquel momento las ideas preconcebidas que se tenían, de cómo procesos altamente definidos eran la mejor forma de obtener software en tiempo, costo y calidad requerida.

Ejemplo de metodologías ágiles más destacadas también lo son la Programación Extrema, XP (eXtreme Progrmming), Scrum, Crystal Clear, DSDM (Dynamic Systems Development Method), ASD (Adaptative Driven Development), XBreed y Extreme Modeling.

3.1 METODOLOGÍA FDD *(FEATURE DRIVEN DEVELOPMENT)*

Peter Coad es considerado uno de los referentes más importantes dentro de la Ingeniería del Software; Coad ha sido uno de los principales pioneros detrás del movimiento de la orientación a objetos y empezó a trabajar con Ed Yourdon a principios de los noventa en una nueva metodología basada en el paradigma de la programación orientada a objetos. El ciclo de vida de vida propuesto por FDD se puede observar en la figura 7 y está compuesta por cinco procesos, dos de los cuales se realizan tantas veces como iteraciones se planifiquen en el desarrollo.

Figura 15. Ciclo de vida FDD según Coad



Fuente: *A practica guide to Feature Driven Development.*

La FDD o Desarrollo Basado en Características se estructura alrededor de la definición de características que representan la funcionalidad que debe tener el sistema y tienen a su vez un alcance lo suficientemente corto y bien definido para que puedan ser implementadas en un par de semanas. FDD también posee una jerarquía de características.

3.2 FUNDAMENTOS DE LA FDD

FDD está construida como buen proceso de desarrollo en base a un núcleo conformado por lo que podría definirse como “buenas prácticas”. La escogencia de dichas prácticas no es en sí lo novedoso sino en particular la forma de combinarlas, que es en este caso lo revolucionario. Cada una de estas prácticas complementa y refuerza las demás. El resultado es un todo mayor que la suma de sus partes, donde la implementación de únicamente una o dos de estas prácticas no conseguirá los mismos beneficios que se obtienen al aplicar el proceso completo que plantea la FDD.

Las buenas y mejores prácticas que conforman la FDD son:

3.2.1 Modelado de objetos del dominio

En esta práctica se diagraman las clases de objetos dentro del problema o dominio y las relaciones entre ellas con UML, incluyendo diagramas de secuencia de alto nivel, los cuales determinan específicamente cómo los objetos interactúan y sus completas responsabilidades

El modelado de objetos del dominio es una forma de descomposición de los objetos donde el problema es dividido en los objetos significantes involucrados

dentro del problema. El diseño y la implementación de cada objeto o clase identificado en el modelo es un problema más pequeño por resolver. Cuando las clases completas son combinadas forman la solución para un problema más grande.

3.2.2 Desarrollo por características

Una vez identificadas las clases en el dominio es posible diseñar e implementar cada una dentro de en una iteración, luego una vez esté completo el conjunto de clases se integrarán y se obtendrá así una parte del sistema.

Cada metodología que incluye algún tipo de proceso que implique la descomposición funcional, la cual descompone problemas de alto nivel en problemas más manejables, usa en sus documentos de especificación funcional modelos de casos, casos de uso y características, todas representando requerimientos funcionales, teniendo cada una de estas representaciones sus ventajas y desventajas.

Tradicionalmente se han siempre tomado los propósitos del sistema y se han dividido en un número de problemas más pequeños, definiendo así un conjunto de subsistemas o módulos para resolver estos problemas más pequeños. Entonces, cada subsistema se divide en una lista jerárquica de requerimientos funcionales. Cuando se tienen estos requerimientos lo suficientemente detallados o de una granularidad determinada se sabe como diseñar e implementar cada uno de ellos, deteniéndose en este punto la descomposición del problema. El proyecto es manejado y guiado a través de funciones, conjuntos de requerimientos funcionales son dados a los desarrolladores para ser implementados, y luego su progreso en esta tarea es medido.

Un problema mayor es que los requerimientos funcionales tienden a mezclarse, funciones de interfaz de usuario, almacenamiento de datos y comunicaciones se mezclan con las funciones de negocio, el resultado, es que los desarrolladores algunas veces gastan grandes cantidades de tiempo de trabajo en características técnicas a expensas de las características de negocio, o sea un proyecto con gran persistencia a los mecanismos pero en el cual sus características de negocio son un fracaso.

Una buena solución a este problema es restringir nuestra lista de requerimientos funcionales a aquellos de valor para el cliente o usuario y asegurar que sean descritos en un lenguaje que el cliente o usuario puedan entender. Estos requerimientos son llamados funciones valoradas por el cliente client-valued functions o características features. Una vez las características para un sistema han sido identificadas, son usadas para guiar el desarrollo en FDD. Entregar un pieza de la infraestructura puede ser importante, incluso critico para el proyecto

pero puede no tener significado para el cliente porque no tiene ningún valor intrínseco para el negocio. Mostrar el progreso en términos de características completadas es algo que el cliente puede entender y darle valor, también los clientes puede priorizar dichas características en términos de su significado para el negocio.

A partir de esto se puede definir que el término característica en FDD es muy especifico, una característica es una función pequeña y valorada por el cliente expresada de la forma <acción> <resultado><objeto> con las proposiciones apropiadas entre la acción el resultado y el objeto.

Además sobre las características se puede decir que:

Las características son pequeñas: La mayoría de características son lo suficientemente pequeñas para ser implementadas en unas pocas horas o días, dos semanas es el límite máximo. Sin embargo las características son más que métodos de acceso que simplemente retornan o asignan el valor de un atributo, cualquier función que sea demasiado compleja para ser implementada en más de dos semanas será descompuesta en funciones más pequeñas hasta que cada subproblema sea tan pequeño para que pueda ser llamado característica.

Las características son valoradas por el cliente: En un sistema de negocio una característica mapea un paso en algunas actividades dentro del proceso de negocio, en otros sistemas una característica es igual a unos pasos u opciones dentro de una tarea que será desarrollada por un usuario. Ejemplos de características son:

1. Calcular el total de una venta.
2. Validar el acceso del usuario.
3. Obtener el balance de la cuenta bancaria.
4. Autorizar una transacción con tarjeta de crédito al portador de la tarjeta.
5. Ejecutar un servicio programado en un carro.

3.2.3 Propietario de clases individuales (código)

La propiedad sobre las clases (código) en un proceso de desarrollo denota quien (persona o rol) es finalmente responsable por los contenidos de una clase (pieza de código). Cada lenguaje de programación popular actual usa el concepto de clase para proveer encapsulación, cada clase define un único concepto o tipo o entidad por lo tanto tiene sentido hacer de las clases los elementos de código más pequeños los cuales se les asignará un propietario. Entonces la propiedad sobre el código se convierte en la propiedad sobre la clase. Esta es la práctica usada dentro de FDD, a los desarrolladores se le asigna propiedad sobre un conjunto de clases del modelo de objetos del dominio.

3.2.4 Equipos de características

Al construir el modelo de objetos de dominio son identificadas las clases clave en el problema dominio, la práctica de asignar permisos a las clases a su vez asigna esas clases a desarrolladores específicos. Para organizar de la mejor manera los propietarios de las clases, al construir la característica, se asegura que haya una sola persona responsable por el desarrollo de cada clase, haciéndolo de la misma forma para las características, se requiere asignar a cada característica un propietario o alguien que vaya a ser el responsable y asegure que esa característica sea desarrollada apropiadamente. La implementación de una característica seguramente involucrará más de una clase y por lo tanto más de un propietario de clases. Entonces el propietario de la característica tendrá que coordinar los esfuerzos de múltiples desarrolladores, esto es un “Team Lead Job” o trabajo de líder de equipo, se escogerán algunos de los mejores desarrolladores para hacerlos los líderes de equipo y asignarles conjuntos de características a ellos.

A los líderes de equipo se les permite escoger los desarrolladores basados en su propia experiencia para trabajar en el equipo de característica, desarrollando aquellas características que involucran aquellas clases. Algunas ideas importantes sobre los equipos de características son:

Un equipo de características dado el pequeño tamaño de las características, se mantiene pequeño, típicamente entre 3 y 6 personas.

Por definición un equipo de características está compuesto por todos los propietarios de las clases, de los cuales se necesita modificar o mejorar una de las clases como parte del desarrollo de una característica en particular.

Cada miembro del equipo de característica contribuye al diseño o implementación de la misma bajo la guía del habilidosos y experimentados desarrollador.

Los programadores en jefe también pueden ser propietarios de clase y tomar parte en equipos e características liderados por otros programadores en jefe, esto ayuda a que los programadores en jefe trabajen en conjunto y se mantengan cerca al código.

3.2.5 Inspecciones

FDD depende en gran medida de las inspecciones para asegurar la calidad de los diseños y del código. Cuando son realizadas correctamente, las inspecciones son enormemente útiles en la mejora de la calidad de los mismos. El propósito principal de las inspecciones es la detección de defectos, sin embargo si son realizadas correctamente darán dos beneficios adicionales:

Transferencia de conocimiento: las inspecciones son métodos para diseminar cultura de desarrollo y experiencia. Al examinar el código de experimentados desarrolladores, teniendo una explicación completa a través del código y las técnicas usadas, los desarrolladores menos experimentados adquieren rápidamente mejores prácticas de codificación.

Conformidad con los estándares: Una vez los desarrolladores se den cuenta que su código no pasará las inspecciones a menos que cumpla los estándares de diseño y código acordados, ellos seguirán muy probablemente los mismos.

Las inspecciones deben hacerse de tal manera que eliminen el miedo a sentirse avergonzado o humillado por parte de los desarrolladores a los que se les realiza la inspección. Todos deben ver que las inspecciones son primeramente una excelente herramienta de depurado y segundamente una gran oportunidad para aprender entre ellos, además los desarrolladores deben entender que las inspecciones no son revisiones de rendimiento personales.

3.2.6 Construcciones regulares programadas

En intervalos regulares, se toma el código de las características desarrolladas y las respectivas clases que la componen y se construye el sistema completo. Algunos equipos construirán semanalmente, otros diariamente y algunos constantemente, eso depende del tamaño del proyecto y el tiempo que puede tomar construir el mismo. Construcciones regulares ayudan a resaltar errores de integración tempranamente, además aseguran que siempre hay un sistema construido a la fecha que puede servir de demostración al cliente, inclusive si solo puede hacer unas simples tareas desde la línea de comandos; al estar desarrollando por características significará que esas simples tareas tienen un valor considerable para el cliente. Un proceso de construcción regular puede ser mejorado con:

- Generando documentación.
- Corriendo scripts métricas y de auditoría en el código para resaltar cualquier área con problemas potenciales y verificar el cumplimiento de los estándares.
- Ser usado como base para realizar pruebas de regresión, verificando que las características se mantengan al agregar características nuevas.
- Crear notas de versión y de construcción del proyecto, agregando las características nuevas, las correcciones de errores, etc.

Los resultados de las mismas pueden ser publicados automáticamente vía intranet para que todo el equipo del proyecto y la organización tengan actualizaciones minuto a minuto.

3.2.7 Manejo de configuraciones

El proyecto debe contar con un sistema de control de versión que identifique y mantenga el código fuente de todas las características que han sido completadas a la fecha y su respectivo histórico de cambios. Además del código, documentación y todo tipo de información perteneciente al proyecto debe encontrarse dentro del sistema de control de versión, inclusive contratos con el cliente y acuerdos de que es lo que se va a desarrollar son candidatos para el control de versión. Cambios en los procesos que están siendo usados y ajustes que deban realizarse durante la construcción del proyecto deberán ser versionados y firmados por los Jefes de Proyecto o Programadores en jefe.

3.2.8 Reporte – visibilidad de los resultados

Teniendo una imagen precisa del estado del proyecto y de que tan rápido el desarrollo agrega nueva funcionalidad al resultado general deseado, da a los líderes la información que ellos necesitan para ajustar el proyecto correctamente. FDD provee un simple y liviano método para obtener información confiable y precisa y sugiere un número de sencillos e intuitivos formatos para reportar el progreso a todos los roles dentro y fuera del proyecto.

3.3 PROCESOS DE LA FDD

La metodología FDD se inicia con la creación de un modelo del objeto del dominio en colaboración con los Expertos del Dominio. Usando la información de la actividad de modelado y de cualquier otra actividad de requerimientos que tenga lugar, los desarrolladores crean la lista de características. Entonces un esbozo del plan es hecho y las responsabilidades son asignadas. Ahora se está preparado para tomar pequeños grupos de características a través de un diseño y construir dentro de las iteraciones respectivas que no deben ser mayores a dos semanas para cada grupo y regularmente mucho más cortas, algunas veces solo unas horas, este proceso de construcción se repite hasta que no hayan más características.

3.3.1 Desarrollo de un modelo del objeto general

La FDD consiste de cinco procesos, figura 15. Para el primer proceso, los miembros del equipo de dominio y desarrollo trabajan juntos bajo la guía de un modelador del objeto del dominio experimentado (Arquitecto en jefe). Los miembros del equipo del dominio presentan una guía inicial de alto nivel del alcance del sistema y su contexto. Entonces los miembros del dominio presentan quías más detalladas de cada una de las áreas del problema. Después de cada una de estas guías los miembros del dominio y desarrolladores trabajan en pequeños grupos para producir modelos para cada una de las áreas del dominio.

Cada pequeño grupo presenta sus resultados al equipo, el equipo compara y somete a discusión los diferentes modelos, finalmente decide cual es el modelo más apropiado para el área del dominio. Los detalles e incluso la forma del modelo general son ajustados a través del desarrollo si es necesario.

3.3.2 Construcción de la lista de características

Construyendo sobre el conocimiento adquirido durante la actividad inicial de modelado, el equipo construye una lista clara de características. Una característica es definida como una función preciada para el cliente expresada en la forma <acción><resultado><objeto>. Documentos sobre requerimientos existentes, tales como casos de o uso o especificaciones funcionales son también utilizados como entrada. El dominio es revisado, usando una división similar a la usada por los expertos del dominio cuando hacen la guía del dominio durante la actividad de modelado. A estas áreas del dominio se les llama también conjuntos principales de características (main feature sets), las características son agrupadas dentro de estos conjuntos principales de características. Un conjunto de características usualmente refleja una actividad particular del negocio. Después de una compilación inicial de características, los usuarios y patrocinadores del sistema revisan la lista de características para validarla y completarla.

3.3.3 Planear por característica

El tercer proceso es ordenar los conjuntos de características o el grupo mayor de características en un plan y asignarlos a los programadores jefe. Lo conjuntos de características son ordenados, dependiendo de la prioridad y dependencias. También, las clases identificadas son asignadas a los desarrolladores individuales. El propietario de una clase es responsable por su desarrollo.

3.3.3 Diseñar y construir por características

Los procesos cuatro y cinco son el desarrollo en sí. El programador en jefe selecciona un pequeño grupo de características para desarrollar en los siguientes días (no más de dos semanas). Él identifica las clases involucradas y los correspondientes propietarios de las clases formando el equipo de característica para esta iteración. Este equipo de característica produce diagramas de secuencia detallados y escribe los encabezados de clases y métodos. Después de una exitosa inspección, los propietarios de las clases añaden el código respectivo a sus clases, hacen pruebas unitarias, integran y mantienen una inspección del código. Una vez el programador en jefe está satisfecho, las características completas son promovidas a la construcción principal, y los procesos 4 y 5 se repiten con los siguientes grupos e características.

4. TRABAJO PRELIMINAR

4.1 PREPARACIÓN DE RECURSOS

En esta sección, se describen el procedimiento necesario que fue realizado para iniciar el desarrollo de este proyecto.

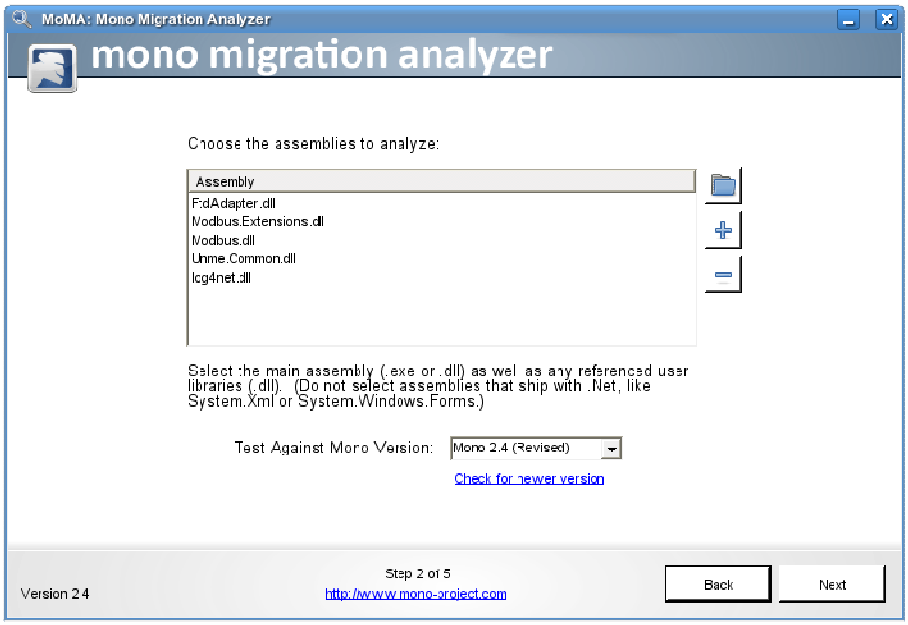
4.1.1 Migración de la librería NModbus 1.8.0 al Framework Mono 2

El proceso de migrar la Biblioteca NModbus 1.8.0, al Framework Mono 2 se realizó mediante los siguientes pasos.

4.1.1.1 Análisis inicial de los ensamblados .Net a través de la utilidad MoMA (Mono Migration Analyzer)

El análisis se lleva a cabo en los cinco pasos que describe esta utilidad. En el paso dos se han incluido los ensamblados FtdAdapter.dll, Log4net.dll, Unme.Common.dll y Modbus.dll para que sean analizados en busca de posibles problemas (ver figura 16). Una lista con la descripción de los posibles problemas se puede encontrar en la descripción de errores de MoMA[7].

Figura 16. Paso dos MoMA

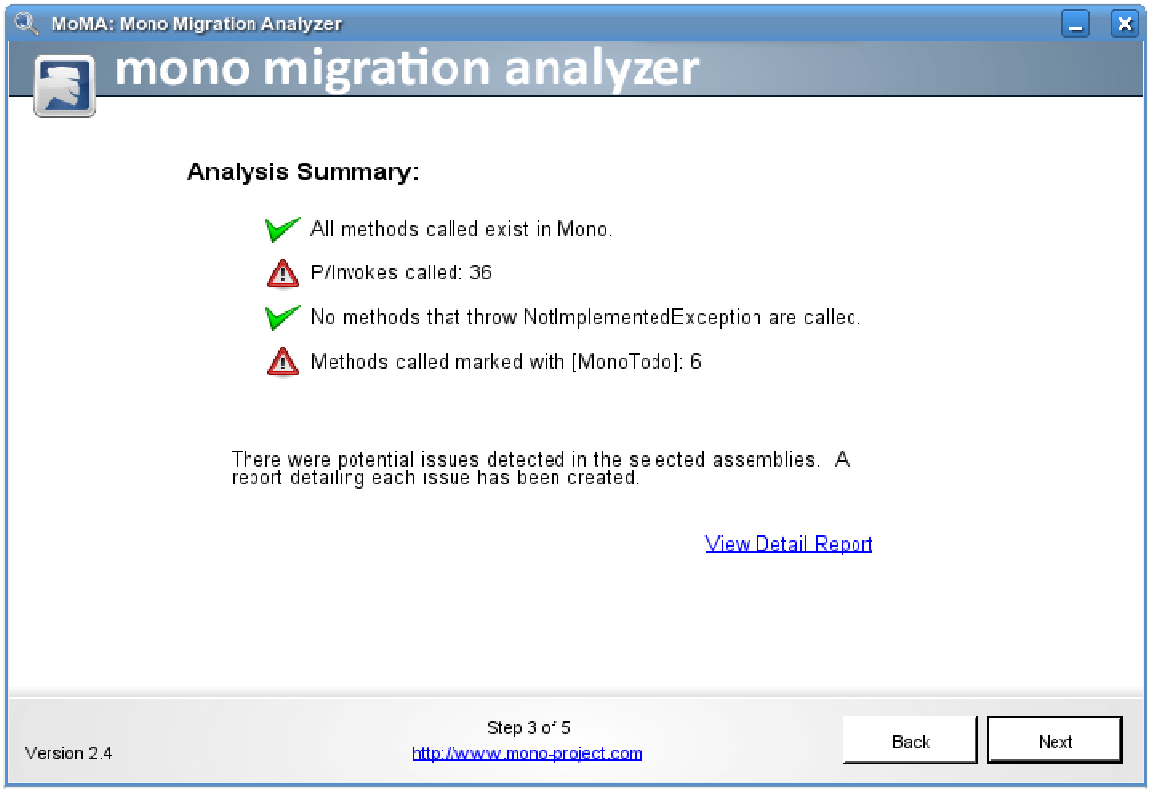


Fuente: *Autores del proyecto*

[7]Mono Migration Analyzer Disponible en: *http://www.mono-project.com/MoMA_-_Issue_Descriptions*

En esta migración se han detectado treinta y seis llamadas a la plataforma (P/Invokes) y seis métodos que están marcados como [MonoTodo] o de implementación pendiente ver figura 17.

Figura 17. Paso tres MoMA



Fuente: *Autores del proyecto*

Después de esto se procede a revisar el reporte detallado el cual indica que las llamadas a la plataforma pertenecen a FtdAdapter.dll y Log4net.dll. Gracias a MoMA se han detectado los ensamblados que pueden generar problemas y en los cuales se enfocarán los siguientes pasos, ver figura 18

.Figura 18. Reporte detallado de la herramienta MoMA

log4net.dll	1.2.10.0	0	0	6	17
Calling Method	**Method with [MonoTodo]**				**Reason**
void ActivateOptions ()	string EventLog.LogNameFromSourceName (string, string)				remote machine is not supported
void ActivateOptions ()	void EventLog.DeleteEventSource (string, string)				remote machine is not supported
void ActivateOptions ()	string EventLog.LogNameFromSourceName (string, string)				remote machine is not supported
void ActivateOptions ()	string EventLog.LogNameFromSourceName (string, string)				remote machine is not supported
void CreateEventSource (string, string, string)	void EventLog.CreateEventSource (EventSourceCreationData)				remote machine is not supported
void Configure (ILoggerRepository, Uri)	ICredentials CredentialCache.get_DefaultCredentials ()				Need EnvironmentPermission implementation first
Calling Method	**P/Invoke Method**				**P/Invoke Library**
void Append (LoggingEvent)	IntPtr ColoredConsoleAppender.GetStdHandle (uint)				Kernel32.dll
void Append (LoggingEvent)	IntPtr ColoredConsoleAppender.GetStdHandle (uint)				Kernel32.dll
void Append (LoggingEvent)	bool ColoredConsoleAppender.GetConsoleScreenBufferInfo (IntPtr, ColoredConsoleAppender/CONSOLE_SCREEN_BUFFER_INFO&)				Kernel32.dll
void Append (LoggingEvent)	bool ColoredConsoleAppender.SetConsoleTextAttribute (IntPtr, UInt16)				Kernel32.dll
void Append (LoggingEvent)	bool ColoredConsoleAppender.SetConsoleTextAttribute (IntPtr, UInt16)				Kernel32.dll
void ActivateOptions ()	int ColoredConsoleAppender.GetConsoleOutputCP ()				Kernel32.dll
void ActivateOptions ()	void LocalSyslogAppender.openlog (IntPtr, int, LocalSyslogAppender/SyslogFacility)				libc
void Append (LoggingEvent)	void LocalSyslogAppender.syslog (int, string, string)				libc
void OnClose ()	void LocalSyslogAppender.closelog ()				libc
void Append (LoggingEvent)	int NetSendAppender.NetMessageBufferSend (string, string, string, string, int)				netapi32.dll
void Append (LoggingEvent)	void OutputDebugStringAppender.OutputDebugString (string)				Kernel32.dll
string GetErrorMessage (int)	int NativeError.FormatMessage (int, IntPtr&, int, int, String&, int, IntPtr)				Kernel32.dll
WindowsIdentity LogonUser (string, string, string)	bool WindowsSecurityContext.LogonUser (string, string, string, int, int, IntPtr&)				advapi32.dll
WindowsIdentity LogonUser (string, string, string)	bool WindowsSecurityContext.DuplicateToken (IntPtr, int, IntPtr&)				advapi32.dll
WindowsIdentity LogonUser (string, string, string)	bool WindowsSecurityContext.CloseHandle (IntPtr)				kernel32.dll
WindowsIdentity LogonUser (string, string, string)	bool WindowsSecurityContext.CloseHandle (IntPtr)				kernel32.dll

Fuente: *Autores del proyecto.*

Por último se llena la información del reporte y se envía a la comunidad que soporta el proyecto Mono para su análisis, esperando que los métodos marcados como [MonoTodo] estén implementados en la próxima actualización.

Si bien MoMA solo puede informar de problemas potenciales, da una idea de las áreas que se deben cubrir a la hora de portar el ensamblado.

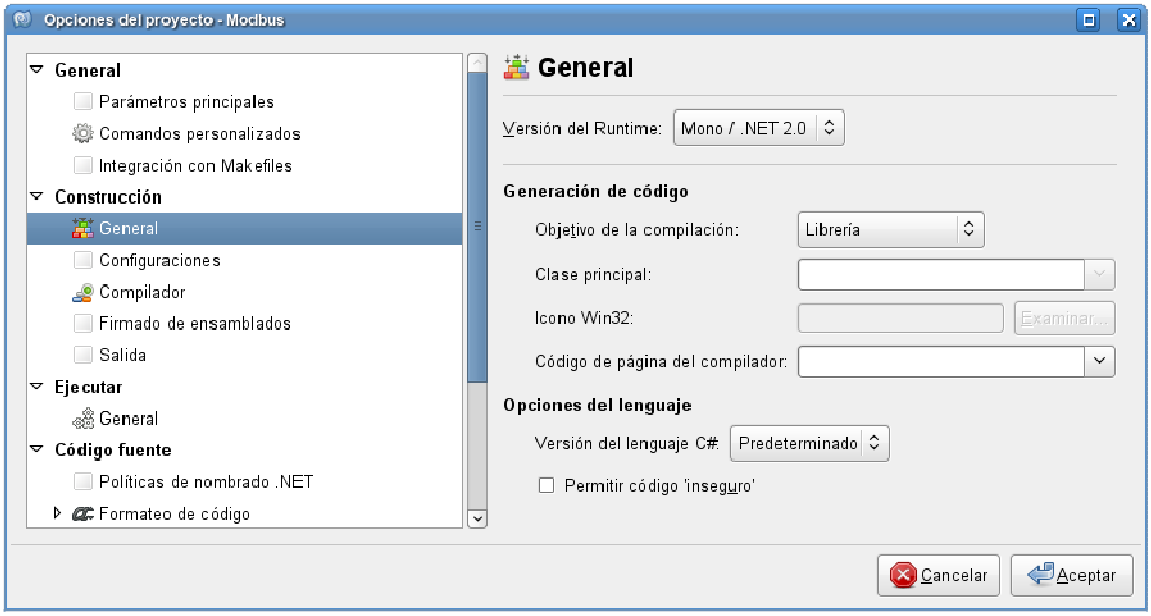
4.1.1.2 Apertura inicial de la solución en MonoDevelop

El formato del proyecto NModbus está actualmente manejado por Visual Studio .Net 2008 en forma de solución, esto permite que MonoDevelop pueda leer la solución sin mayores inconvenientes, ya que MonoDevelop cuenta con soporte de soluciones y proyectos creados con las últimas versiones de Visual Studio .Net.

4.1.1.3 Selección del Framework y primer intento de compilación

En la vista de la solución se ubica el proyecto Modbus, al hacer clic derecho sobre el proyecto aparece el menú contextual, se selecciona “Opciones” para acceder a las propiedades del mismo. Ya en la ventana de opciones se selecciona el árbol “construcción”, y en la opción general se selecciona Mono /.NET 2.0, ver figura 19. El proceso se repite para todos los proyectos que componen la solución, dejando entonces a Mono el proceso de construcción de la misma.

Figura 19. Selección del Framework y primer intento de compilación

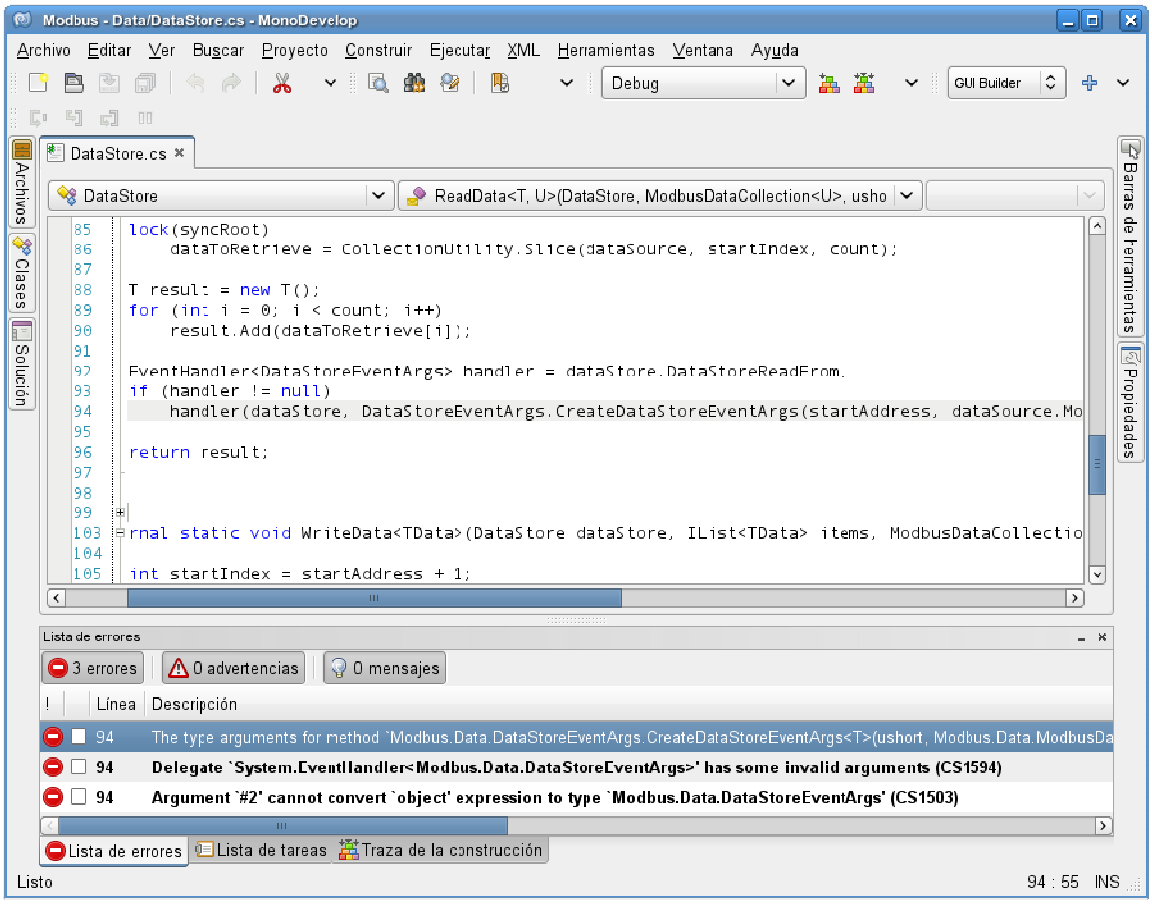


Fuente: *Autores del proyecto.*

4.1.1.4 Corrección de errores de compilación y llamadas a la plataforma (P/Invokes)

En el primero intento de generar la solución se detecta un error de compilación inesperado que al parecer se debe a un parámetro enviado al método CreateDataStoreEventArgs, el cual no se encuentra fuertemente tipado, ver figura 20.

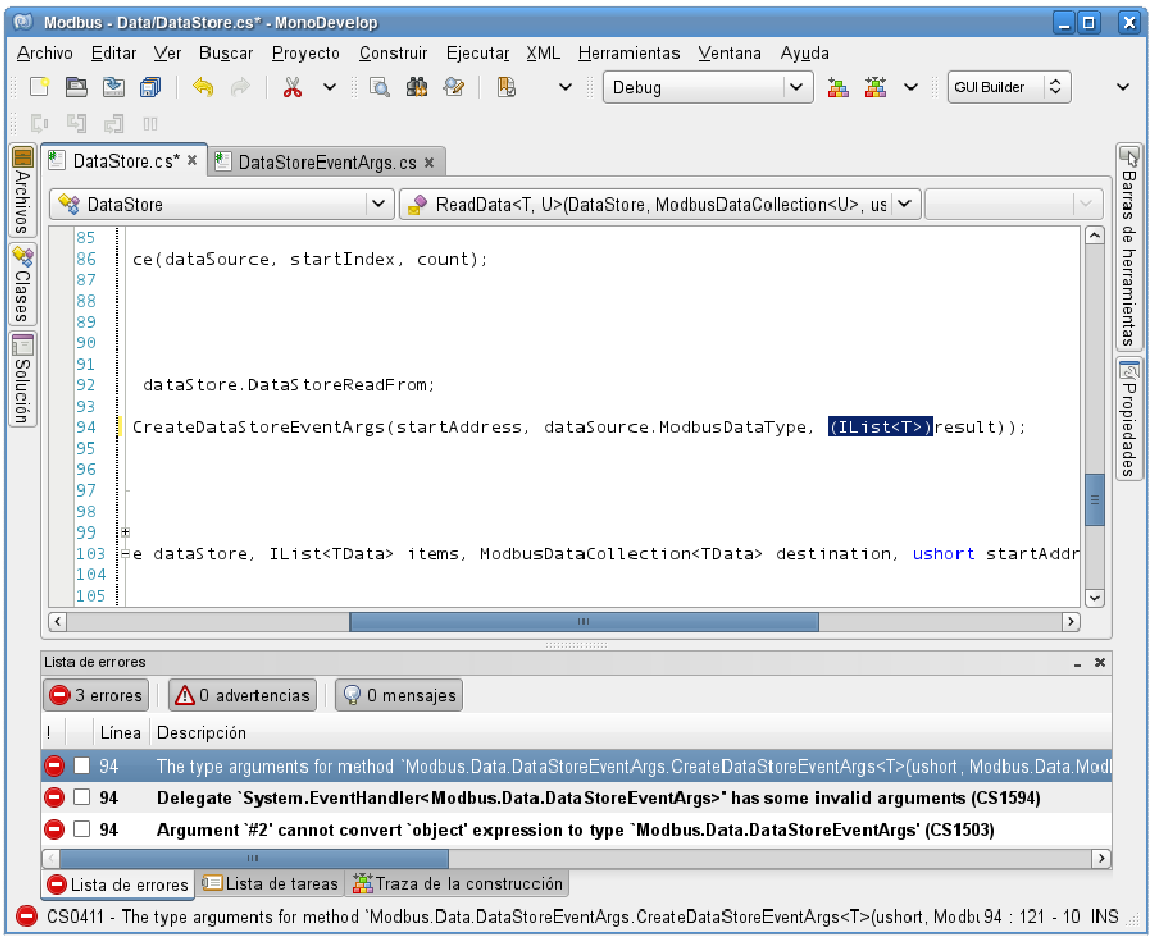
Figura 20. Errores de compilación NModbus.sln



Fuente: *Autores del proyecto*

La línea de código 94 del archivo DataStore.cs es corregida agregando un casting implícito antes del parámetro “result” al tipo“(IList<T>)”, ver figura 21, esto indica al compilador el tipo y soluciona el problema.

Figura 21. Corrección línea de código 94 del archivo DataStore.cs



Fuente: *Autores del proyecto.*

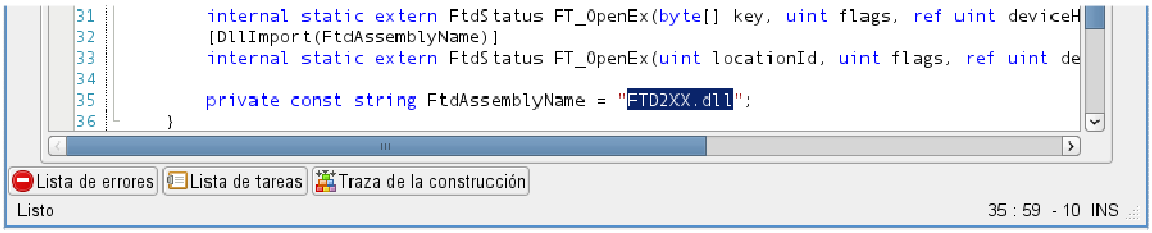
Después de esto hay que tratar los P/Invokes. Para la biblioteca Log4Net.dll es muy sencillo puesto que a la fecha tiene una versión compilada para Mono 2.0 la cual se puede descargar del sitio oficial del proyecto[8]. Basta con reemplazar la biblioteca con su respectiva versión para Mono, la cual se encuentra en el directorio source/lib sobre la raíz del proyecto, y compilar nuevamente.

Al analizar la Biblioteca FtdAdapter.dll dentro del árbol de la solución, se observa que es una interfaz para la Biblioteca FTD2XX.dll la cual es provista por Future Technology Devices International LTDA, figura 22. Debido principalmente al manejo que se le da a los dispositivos USB en los sistemas operativos Windows los problemas de las llamadas a la plataforma (P/invokes) realizados por esta

[8] Sitio oficial Apache log4net.dll *http://logging.apache.org/log4net/index.html*

biblioteca impiden que la característica de comunicación USB sea por el momento multiplataforma, y realizar un cambio sustancial en la Biblioteca NModbus para agregar soporte USB a sistemas operativos UNIX no es parte de los alcances del proyecto.

Figura 22. Biblioteca FTD2XX.dll



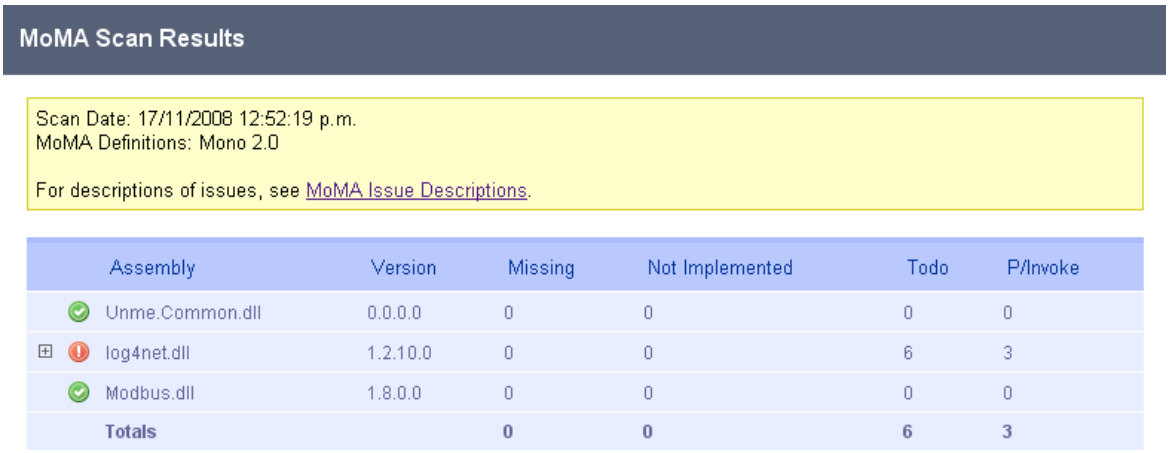
Fuente: *Autores del proyecto.*

Además el proyecto no hace uso de la interfaz USB por tanto se procede a eliminar el soporte sobre la misma. El primer paso es eliminar de la solución el Proyecto FtdAdapter, seguido de esto se procede a eliminar todas las referencias al proyecto junto con el código asociado en los ejemplos y en las pruebas. También se deben eliminar del proyecto Modbus.Integration.Tests los archivos con extensión .cs que contienen pruebas referentes a la interfaz USB tales como NmodbusUsbRtuMasterFixture.cs. Después de esto se puede realizar la compilación.

4.1.1.5 Análisis final de los ensamblados resultado de la compilación

Finalmente se va a llevar a cabo un análisis con MoMA, de la misma forma, descrita en el numeral 4.1.1.1. En esta ocasión solo se han encontrado tres llamadas a la plataforma (P/Invokes) y los mismos 6 [MonoTodo]. Respecto a las llamadas a la plataforma, estas son generadas por Log4net.dll, este ensamblado está compilado para Mono 2 por lo tanto estas llamadas son controladas de manera adecuada.

Figura 23. Reporte final MoMA



MoMA Scan Results

Scan Date: 17/11/2008 12:52:19 p.m.
MoMA Definitions: Mono 2.0

For descriptions of issues, see MoMA Issue Descriptions.

Assembly	Version	Missing	Not Implemented	Todo	P/Invoke
Unme.Common.dll	0.0.0.0	0	0	0	0
log4net.dll	1.2.10.0	0	0	6	3
Modbus.dll	1.8.0.0	0	0	0	0
Totals		**0**	**0**	**6**	**3**

Fuente: *Autores del proyecto.*

5. DESARROLLO DEL PROYECTO

A continuación se presentan cada una de los procesos descritos por la metodología FDD, cada uno se muestra de acuerdo a las plantillas sugeridas por la metodología para estandarizar el registro de tareas y la verificación de las mismas.

En el primer proceso denominado Desarrollo del Modelo del Objeto General, se realiza la abstracción conceptual del modelo de operación, mediante las instrucciones del experto en el dominio a abordar.

La segunda etapa o Construcción de la Lista de Características, como su nombre lo indica es la que el proceso en el que se identifican cada una de las funciones que debe tener el sistema y son significativas para el cliente o usuario final.

El tercer proceso la Planeación por Características se realiza la planeación detallada del proceso de desarrollo, en especial se hace una estimación del tiempo necesario para implementar cada una de las características planteadas en el proceso anterior.

Finalmente el proceso cuatro y cinco, Diseñar y Construir por Característica, es aquel en el que se realiza el diseño detallado de clases y métodos de la implementación y la construcción de los mismos para cada una de las características, este proceso es iterativo y se repite un número de veces igual al número de características existente. Dentro de este proceso se llevan a cabo pruebas unitarias y promoción a construcción del código fuente.

5.1 DESARROLLO DEL MODELO DEL OBJETO GENERAL

5.1.1 Plantilla tareas primer proceso de la FDD

FDD Proceso 1: Desarrollo del Modelo Global

Criterio de entrada

Los Expertos en el Dominio, Programadores Jefe y Arquitecto Jefe han sido seleccionados para el proyecto.

Tareas

Formación del equipo de modelado	**Director de proyecto**	**Requerido**
El equipo de modelado consiste de miembros permanentes del dominio y áreas de desarrollo, específicamente, el Experto en el Dominio y los Programadores Jefe.		

Desarrollar una guía del dominio	Equipo de Modelado	Requerido
El Experto en el Dominio da una visión general del área del dominio a ser modelada. Esto debe incluir información relacionada al área del dominio pero no necesariamente a la implementación.		

Documentos de Estudio	Equipo de Modelado	Opcional
El equipo estudia la documentación disponible del estándar MODBUS y las referencias bibliográficas al mismo.		

Desarrollo del modelo	Equipo de Modelado	Requerido
Los miembros del grupo presentan el modelo propuesto por cada uno para el área de dominio. El arquitecto jefe también puede proponer otras alternativas de modelos. El equipo de modelado selecciona uno de los modelos propuestos o compone un modelo a partir de las ideas de los modelos propuestos.		

Refinar el Modelo del Objeto General	Arquitecto Jefe, Equipo de Modelado	Requerido
Ocasionalmente, el equipo actualiza el modelo del objeto general con las nuevas formas del modelo producidas por las iteraciones de las dos tareas previas. Programadores Jefe		

Notas Escritas del Modelo	Arquitecto Jefe, Programadores Jefe	Requerido
Notas sobre formas detalladas o complejas de los modelos y sobre alternativas importantes son hechas para futuras referencias del proyecto.		

Verificación

Evaluación interna y externa	Equipo de Modelado, Experto en el dominio	Requerido
El Experto en el Dominio participando activamente en el proceso provee evaluación interna o auto evaluación. La evaluación externa es hecha sobre una base de necesidades que se refieren a los usuarios del negocio para la ratificación y clarificación de asuntos que afecten al modelo.		

Criterio de Salida

Para salir de este proceso, el equipo de modelado debe producir un modelo del objeto que satisfaga al Arquitecto Jefe. El modelo del objeto consiste de:

- Diagrama de clases enfocado a la forma del modelo, las clases en el dominio, y como están conectadas las una a las otras y bajo que restricciones, más cualquier operación y atributos identificados.
- Notas que contienen porque una forma particular del modelo fue escogida y/o que alternativas fueron consideradas.

5.1.2 Desarrollo tareas plantilla primer proceso de la FDD

5.1.2.1 Formación del equipo de desarrollo

El equipo de desarrollo queda compuesto por las siguientes personas.

Tabla 5. Integrantes equipo de desarrollo

Rol	Descripción	Personal Asignado
Directores del Proyecto	El director de proyecto es el líder administrativo del proyecto, responsable del reporte del progreso; manejo del presupuesto, equipo, espacio y recursos.	Profesor: Fernando Rojas Profesor: Pedro Trujillo
Experto en el Dominio	Los expertos en el dominio usan su profundo conocimiento del área para explicar a los desarrolladores en varios niveles de detalle las tareas que el sistema debe desarrollar y qué tipo de uso se le dará al sistema.	Ing. José Nikolai Ortiz
Programador en Jefe	Desarrollador o desarrolladores experimentados que han atravesado por completo el ciclo de vida de desarrollo de software. Participan en el análisis y diseño	Angel Andres Vargas Esteban
Propietarios de Clases	Desarrolladores que trabajan como miembros del equipo de desarrollo bajo la guía del Programador en Jefe para diseñar, codificar, probar y documentar las características requeridas por la aplicación.	Angel Andres Vargas Esteban Carol Liset Jaimes Vega
Arquitecto en Jefe	Es el responsable por el diseño general del sistema. Aunque el Arquitecto Jefe tiene la última palabra, en FDD, él es el responsable por las sesiones de de diseño donde el equipo colabora en el diseño del sistema.	Profesor: Fernando Rojas
Modeladores	Desarrolladores que trabajan en conjunto con los Expertos en el Dominio y el Programador en jefe, para formar los equipos de modelado. Estos equipos construyen y presentan modelos del objeto del dominio.	Angel Andres Vargas Esteban Carol Liset Jaimes Vega
Programadores	Desarrolladores que implementan las funciones del sistema.	Angel Andres Vargas Esteban Carol Liset Jaimes Vega

Fuente: *Autores del proyecto.*

5.1.2.2 Desarrollo de la guía del dominio

Para este proyecto, la Guía del Dominio está estructurada como una lista de debes, clasificados principalmente en requerimientos operativos y requerimientos de interfaz. A continuación se presenta la lista obtenida a través de varias reuniones con el Experto en el Dominio.

Requerimientos Operativos

1. Permitir seleccionar entre modo Master o Slave.
2. Permitir configurar el modo de comunicación (ASCII,RTU,TCP).
3. Para los modos ASCII y RTU permitir configurar el puerto serial, baud rate,bits de datos, paridad y bits de parada.
4. Para el modo TCP permitir configurar la dirección IP y el puerto del dispositivo a conectarse (Slave).
5. En modo Master permitir realizar consultas a un dispositivo Modbus en un registro o grupo de registros determinados por el usuario.
6. En modo Master permitir realizar consultas a un dispositivo Modbus en un registro o grupo de registros determinados por el usuario, de manera continua y separadas por un intervalo de tiempo definido por el usuario.
7. En modo Master permitir la escritura a un dispositivo Modbus de un registro con un valor determinado por el usuario.
8. En modo Master permitir la escritura a un dispositivo Modbus de un registro con un valor generado aleatoria mente dentro de un rango determinado por el usuario.
9. En modo Master permitir la escritura a un dispositivo Modbus de un registro con un valor generado aleatoriamente dentro de un rango determinado por el usuario, de manera continua y separada por un intervalo de tiempo definido por el usuario.
10. Todas las consultas y Escrituras deben tener un tiempo máximo de espera de respuesta (Timeout) el cual puede ser configurado por el usuario.
11. En modo Slave la aplicación debe permitir escuchar y responder consultas Modbus en un registro o grupo de registros determinados por el usuario, con un valor determinado por el usuario.
12. En modo Slave la aplicación debe permitir escuchar y responder consultas Modbus en un registro o grupo de registros determinados por el usuario, con un valor generado aleatoria mente dentro de un rango determinado por el usuario.
13. En ambos modos de operación la aplicación interpretara y usar los códigos de errores Modbus de Modicon.
14. En cualquier modo de operación al finalizar la aplicación el simulador mantendrá la configuración actual, del tipo y modo de comunicación.

15. Llevar una lista de dispositivos activos, la cual permite agregar y eliminar dispositivos.

Requerimientos de Interfaz

1. La aplicación contará con una ventana de visualización de la respuesta la cual solo mostrará el resultado de la última consulta.
2. En modo Master para lectura y escritura se debe visualizar: la respuesta, el tipo de respuesta (tipo de dato), su representación en bytes, el tiempo de respuesta.
3. La aplicación contará con una ventana de registros (Logs) donde se podrá visualizar como registros de un solo renglón las consultas realizadas en modo Master durante la sesión de comunicación, es decir durante la ejecución de la aplicación en modo Master.
4. La aplicación contará con una ventana de registros (Logs) donde se podrá visualizar como registros de un solo renglón las consultas recibidas y respuestas realizadas en modo Slave durante la sesión de comunicación, es decir durante la ejecución de la aplicación en modo Slave.
5. En modo Slave la aplicación contará con una ventana donde se visualizan los registros configurados actualmente con valores de respuesta, para los demás registros sin valores asignados, el valor por defecto será cero.
6. Todas las ventanas de registros de simulación podrán ser limpiadas por el usuario cuando lo desee a través del botón correspondiente.
7. La aplicación contará para ambos modos con el botón correspondiente para exportar el registro de la sesión de comunicación en formato CSV.
8. Las ventanas de registro de sesión de comunicación (Logs) serán ventanas independientes de la ventana principal para facilitar su ubicación en cualquier región de la pantalla.
9. En cualquier modo de operación existirá un botón que permitirá volver a los valores por defecto del simulador.
10. Los tipos de datos a visualizar son enteros, hexadecimal y su representación en bits.
11. La lista de dispositivos Slaves activos se debe representar por medio de una tabla que contenga los siguientes campos: Slave id, dirección Modbus inicial, dirección Modbus final, valor de respuesta, y si es escritura o lectura dado que este operando en modo Master.
12. La tabla debe tener un menú contextual que se visualiza al presionar el botón derecho del ratón sobre valor de respuesta, para cambiar la visualización de tipo de dato a cualquiera de los tipos contemplados anteriormente.
13. La aplicación contará con un panel de información General que visualiza el modo actual de comunicación, el tipo de comunicación, tipo y versión de Sistema Operativo y el botón de inicio y parada de operación.

14. Se contará con un formulario para agregar dispositivos a la tabla, en el cual se configuran las diferentes opciones contempladas anteriormente.
15. Se contará con diálogos para la configuración de puertos de comunicación serial, TCP, y de configuración Master y Slave.
16. En el evento de respuesta a una consulta, la fila o registro correspondiente al dispositivo consultado cambiará su color de fuente a azul. Al finalizar la respuesta el color cambiara a negro nuevamente.
17. En el evento de escritura en un registro de un esclavo, la fila o registro correspondiente al dispositivo consultado cambiará su color de fuente a rojo.

5.1.2.3 Documentos de estudio

Toda la documentación estudiada sobre el estándar Modbus, las herramientas y técnicas utilizadas en el desarrollo de este proyecto estás debidamente plasmadas en el marco de referencia del mismo.

5.1.2.4 Desarrollo del modelo

El desarrollo del Modelo del Dominio inicia aplicando una de las tres estrategias para encontrar clases conceptuales. Esto es, hallar las clases conceptuales con la identificación de sustantivos en las frases. Esta es una técnica muy útil debido a su simplicidad la cual fue sugerida por Abbott, R. como Análisis Lingüístico[9]. Esta identifica los sustantivos y las frases sustantivo en la descripción textual del dominio y los considera clases candidatas o atributos. Lista inicial de sustantivos:

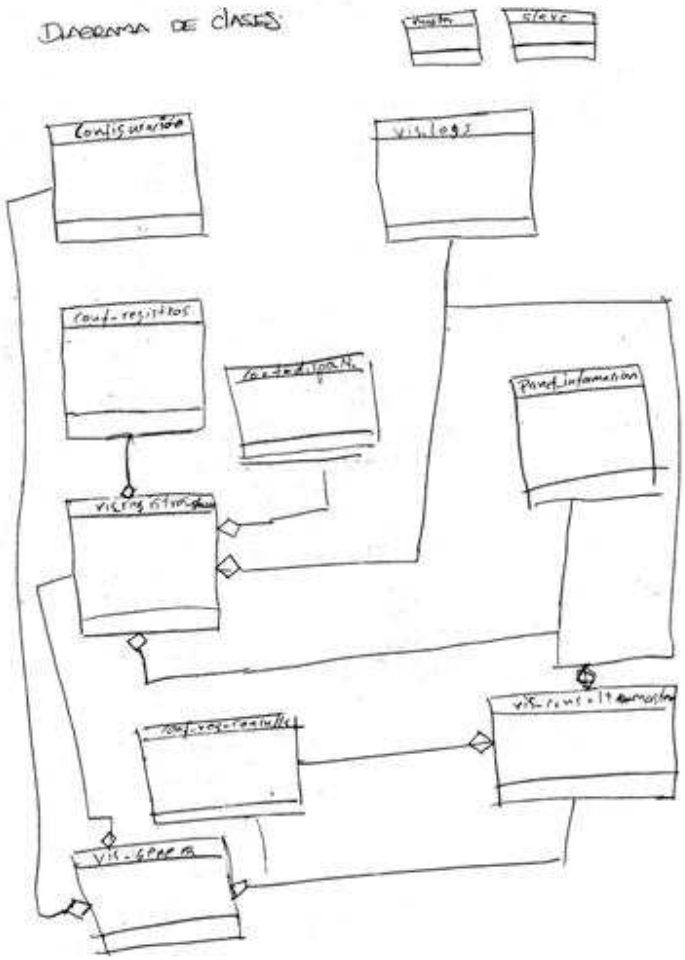
- Master
- Slave
- Modo de Comunicación
- Puerto serial
- Baud rate
- Bits de datos
- Paridad
- Bit de parada
- Dirección IP
- ID del dispositivo
- Consultas
- Dispositivo
- Registros
- Usuario
- Ventana
- Respuesta
- Aplicación
- Modo de operación
- Configuración
- Simulación

A partir de esta lista inicial de sustantivos surgieron por parte de los miembros del equipo de modelado diferentes alternativas en el diseño del Modelo del Dominio. Los primeros diseños eran simples esbozos en papel a partir de los cuales fueron

[9] Técnica sugerida por Abbott, R. En su publicación *Program Design by Informal Descriptions.* Communications of ACM vol 26, 1983.

evolucionando diagramas más completos y refinados sobre el Modelo del Dominio, en la figura 24 se muestra uno de los primeros modelos realizados por el equipo. A continuación se muestran las diferentes etapas que tuvo este proceso de y los resultados parciales que se iban obteniendo.

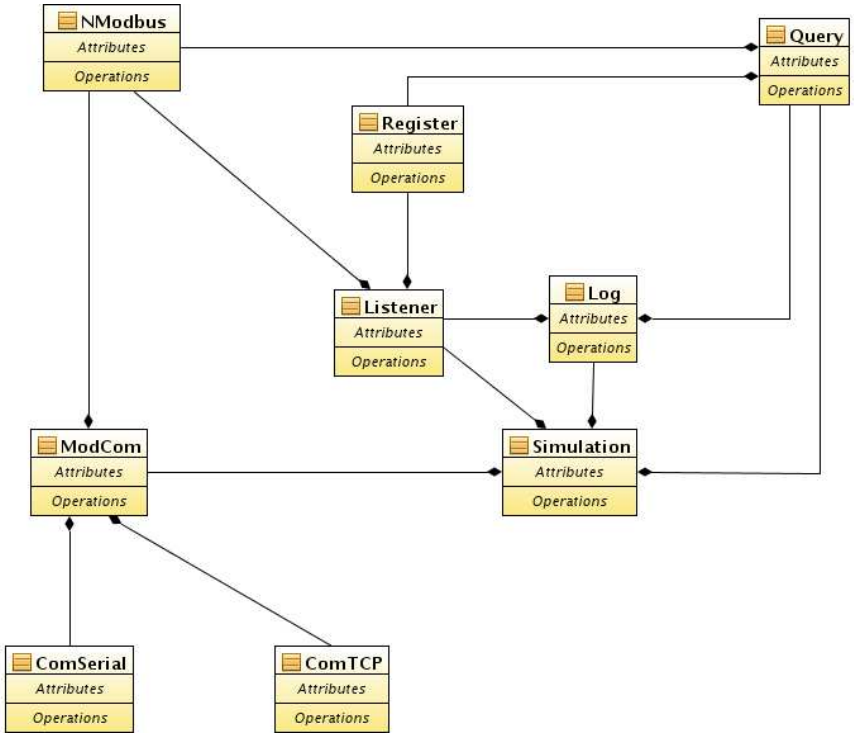
Figura 24. Primer esquema del modelo del dominio



Fuente: *Autores del proyecto.*

Después de los primeros diagramas se llega a un acuerdo en un Modelo del Dominio, este contempla la abstracción de los conceptos más importantes sobre comunicación con Modbus y las relaciones existentes entre dichos objetos. En la figura 25 se presenta un Modelo del Dominio inicial, considerado como completo para la aplicación.

Figura 25. Modelo del dominio primera etapa de modelado



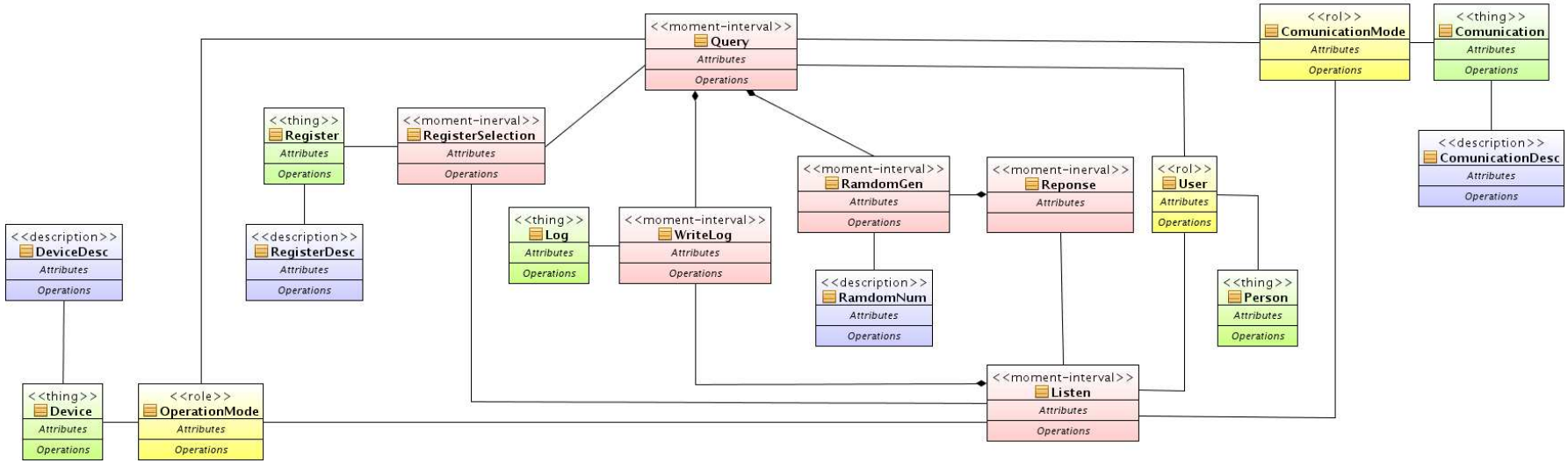
Fuente: *Autores del proyecto*

Este Modelo del Dominio contempla la interacción de la librería NModbus con las clases de la aplicación, sin embargo en análisis posteriores se identifica esto como una falla, ya que de acuerdo a la FDD y en general a los conceptos sobre construcción de un Modelo del Dominio, este tipo de consideraciones tan específicas de la implementación, no deben tenerse en cuenta durante esta etapa.

El siguiente y modelo definitivo antes de la tarea de refinamiento, está basado en la técnica de Modelado en Color[10] sobre la cual está apoyada la FDD. Es importante anotar que aunque en la FDD el Modelo del Dominio está enfocado a negocio, en este caso el Modelo del Dominio, es llamado el Modelo de Operación y describe el funcionamiento de una comunicación basada en protocolo Modbus. En la figura 26 se presenta el Modelo del Dominio o para este caso el Modelo de Operación, con la respectiva clasificación por arquetipos y colores sugerida en la FDD.

[10] Ténica descrita por Peter Coad en su libro, *Java Modeling In Color With UML,* Peter Coad, Eric Lefecvre, and Jeff de Luca, June 1999.

Figura 26. Modelo del dominio o modelo de operación

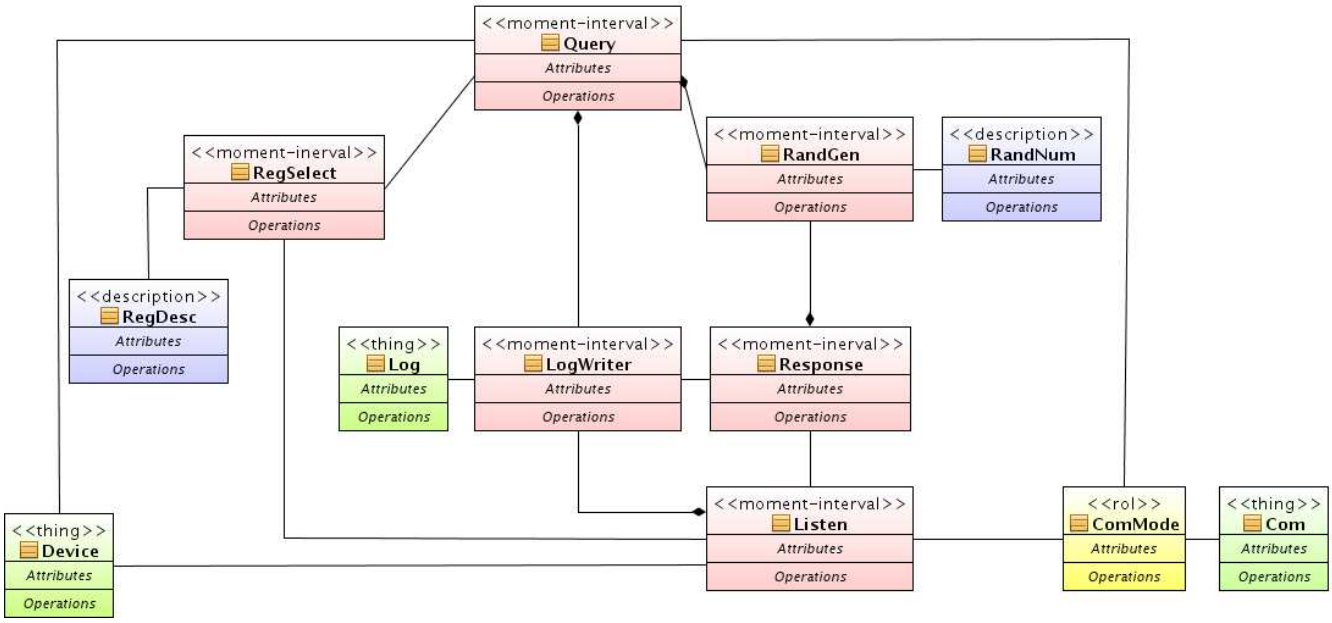


Fuente: *Autores del proyecto.*

5.1.2.5 Refinar el modelo de objeto general

Finalmente después de varias sesiones de verificación con el Experto en el Dominio y de consideraciones de diseño, se obtiene un Modelo del Dominio o Modelo de Operación final y refinado, este es utilizado junto con la Guía del Dominio para la realización de la Lista de Características en el siguiente proceso planteado por la FDD. En la figura 27 se puede observar el Modelo del Dominio final

Figura 27. Modelo del operación refinado



Fuente: *Autores del proyecto.*

5.1.2.6 Notas escritas del modelo

Junto con el desarrollo del Modelo del Dominio y como parte de la descripción de las características de las clases involucradas en el mismo, se realiza un diccionario de clases el cual permite una mejor interpretación del Modelo del Dominio y sirve como herramienta de apoyo en las siguientes etapas de diseño y construcción por característica.

Nombre	Device
Descripción	Clase que contiene la información básica para representar e identificar un dispositivo Modbus, ya sea Master o Slave.
Asociaciones	Query, Listen

Nombre	RegDesc
Descripción	Clase que implementa el tipo o formato de los datos y su presentación.
Asociaciones	RegSelect, RecDesc

Nombre	Log
Descripción	Clase que implementa los métodos para registrar los sucesos ocurridos en la sesión de comunicación. Tales como inicio de escucha, consulta o respuestas.
Asociaciones	WriteLog

Nombre	Com
Descripción	Clase que contiene los atributos y métodos para manejar los tres modos de comunicación TCP, Serial ASCII, Serial RTU.
Asociaciones	ComMode

Nombre	RandNum
Descripción	Contiene los métodos para generar número aleatorios.
Asociaciones	RandGen

Nombre	RegSelect
Descripción	Contiene los registro seleccionados para las consultas o sobre los cuales se genera una respuesta dependiendo del modo de operación.
Asociaciones	Query, Reg, Listen

Nombre	LogWriter
Descripción	Clase que contiene lo que se lleva registrado de la sesión de comunicación actual.
Asociaciones	Query, Listen, Log

Nombre	RandGen
Descripción	Contiene la información del intervalo entre el cual se genera el número aleatorio.
Asociaciones	Query, RandNum, Reponse

Nombre	Response
Descripción	Genera respuestas con base a las consultas realizadas en modo Slave.
Asociaciones	RandGen, Listen

Nombre	Query
Descripción	Realiza las consultas Modbus, con los parámetros especificados en modo Master a través de las clases asociadas y basadas en la documentación específica.
Asociaciones	RandGen, Listen

Nombre	Listen
Descripción	Escucha consultas realizadas por dispositivos Modbus Master con los parámetros específicados a través de las clases asociadas y basadas en la documentación específica.
Asociaciones	Device, RegSelect, Response, ComMode

Nombre	ComMode
Descripción	Contiene los atributos de la representación del modo de comunicación actual.
Asociaciones	Query, Listen, Com

5.2 CONSTRUCCIÓN DE LA LISTA DE CARACTERÍSTICAS

5.2.1 Plantilla segundo proceso de la FDD

Criterio de entrada

El equipo de modelado ha completado exitosamente el proceso 1 de la FDD. Desarrollo del Modelo General.

Tareas

Formación del equipo de Lista de Características	Director de proyecto, Director de Desarrollo	Requerido
El equipo de Lista de Características comprende los Programadores Jefe del equipo de modelado en el proceso 1.		

Construcción de la Lista de Características	Equipo de Lista de Características	Requerido
Esta tarea es una simple descomposición funcional, iniciando con el particionamiento del dominio usado por los Expertos del Dominio para sus guías del área de dominio en el proceso 1 de la FDD. El dominio es descompuesto en *áreas (conjuntos de características mayores)* que comprenden *actividades (conjuntos de características)* que comprenden a su vez *características*, cada uno de las cuales representa un paso en una actividad.		

Verificación

Evaluación interna y externa	Equipo de Lista de Características	Requerido
Miembros del equipo de modelado participando activamente en el proceso proveen autoevaluación o evaluación interna. Respecto a la evaluación externa, esta es realizada en base a las necesidades a que hacen referencia los Expertos del Dominio a partir del equipo de modelado o los usuarios, para la ratificación o aclaración de asuntos que afecten la lista de características.		

Criterio de Salida

Para salir de este proceso, el equipo de Lista de Características debe producir, la lista de características a satisfacción del Director de Proyecto. La lista de características consiste de:

- Una lista mayor de características (área).
- Para cada lista mayor de características, una lista de conjuntos de características (actividades).
- Una lista de características para cada conjunto de características (actividad), cada una representando un paso en la actividad de ese conjunto de características.

5.2.2 Desarrollo tareas plantilla segundo proceso de la FDD

5.2.2.1 Formación del equipo de lista de características

De acuerdo a los parámetros estipulados en la FDD, el equipo de Lista de Características, está formado por los programadores jefe del proceso 1. Debido a la naturaleza de este proyecto, y conforme a lo estipulado en la formación del Equipo de Modelado en el proceso 1. El programador en Jefe será el componente principal del equipo de Características, sin embargo el equipo de desarrollo en su totalidad estará involucrado en la adecuada construcción de la Lista de Características.

Programador en Jefe	**Angel Andrés Vargas Esteban**
Desarrolladores	Carol Liset Jaimes Vega

5.2.2.2 Construcción de la lista de características

La construcción de la Lista de Características está basada en la descomposición funcional del problema del dominio y está agrupada en funciones con valor para el cliente. En este caso el Equipo de Lista de Características las agrupó dentro de tres Main Feature Sets, MFS (Conjuntos Principales de Características), el primer MFS es Realizar Simulación Master, el segundo MFS es Realizar Simulación Slave y finalmente el tercer MFS es Características GUI.

A su vez, el MFS Realizar Simulación Master está conformado por dos Feature Sets, FS (Conjunto de Características), estos son Realizar Simulación Master TCP y Master ASCII / Master RTU. De manera análoga el MFS Realizar Simulación Slave está compuesto de los FS Realizar Simulación Slave TCP y Realizar Simulación Slave ASCII / Slave RTU.

Teniendo en cuenta lo anterior la Lista de Características, agrupada y jerarquizada es la siguiente.

Main Feature Set 1: Realizar Simulación Master

Feature Set 1: Realizar Simulación Master TCP

1. Seleccionar el modo de operación Master y tiempo de espera máximo de respuesta, para el objeto Device.
2. Seleccionar tipo de comunicación TCP en el objeto Com.
3. Asignar la dirección IP, y el puerto del dispositivo para configurar ComMode.

4. Realizar consulta Modbus con NModbus y almacenar resultados en Query.
5. Definir un intervalo de tiempo y realizar consultas Modbus automáticas separadas por dicho intervalo y almacenar los resultados en Query.
6. Escribir un registro Modbus con NModbus en un dispositivo esclavo externo y almacenar los resultados en Query.
7. Definir un rango para generar un número aleatorio, generarlo y escribir el número generado en un registro Modbus de un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.
8. Definir un intervalo de tiempo y realizar escrituras Modbus automáticas de números generados aleatoriamente en un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.
9. Visualizar respuesta, tipo de respuesta, su representación en HEX y el tiempo de respuesta de la última consulta en la ventana de Simulación en modo Master.
10. Agregar un renglón con una cadena que contiene la consulta y la respuesta para visualizar la consulta y su resultado en la ventana de Log.

Feature Set 2: Realizar Simulación Master ASCII / Master RTU

1. Seleccionar el modo de operación Master y tiempo de espera máximo de respuesta, para el objeto Device.
2. Seleccionar tipo de comunicación RTU o ASCII en el objeto Com.
3. Asignar el puerto serial, baud rate, bits de datos, paridad y bits de parada para configurar ComMode.
4. Realizar consulta Modbus con NModbus y almacenar resultados en Query.
5. Definir un intervalo de tiempo y realizar consultas Modbus automáticas separadas por dicho intervalo y almacenar los resultados en Query.
6. Escribir un registro Modbus con NModbus en un dispositivo esclavo externo y almacenar los resultados en Query.
7. Definir un rango para generar un número aleatorio, generarlo y escribir el número generado en un registro Modbus de un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.
8. Definir un intervalo de tiempo y realizar escrituras Modbus automáticas de números generados aleatoriamente en un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.

9. Visualizar respuesta, tipo de respuesta, su representación en HEX y el tiempo de respuesta de la última consulta en la ventana de Simulación en modo Master.
10. Almacenar la consulta y su resultado en Log.

Main Feature Set 2: Realizar Simulación Slave

Feature Set 1: Realizar simulación Slave TCP

1. Seleccionar el modo de operación Slave y tiempo de espera máximo de respuesta, para el objeto Device.
2. Seleccionar tipo de comunicación TCP en el objeto Com.
3. Asignar la dirección IP, y el puerto del dispositivo para configurar ComMode.
4. Escuchar y responder consultas en un registro o grupo de registros con un valor determinado por el usuario con Listen.
5. Escuchar y responder consultas en un registro o grupo de registros con un valor generado aleatoriamente dentro de un rango determinado por el usuario.
6. Almacenar la respuesta en Log.

Feature Set 2: Realizar simulación Slave ASCII / Slave RTU

1. Seleccionar el modo de operación Slave y tiempo de espera máximo de respuesta, para el objeto Device.
2. Seleccionar tipo de comunicación ASCII o RTU en el objeto Com.
3. Asignar el puerto serial, baud rate, bits de datos, paridad y bits de parada para configurar ComMode.
4. Escuchar y responder consultas en un registro o grupo de registros con un valor determinado por el usuario con Listen.
5. Escuchar y responder consultas en un registro o grupo de registros con un valor generado aleatoriamente dentro de un rango determinado por el usuario.
6. Almacenar la respuesta en Log.

Main Feature Set 3: Características GUI

Feature Set 1: Características GUI

1. Mostrar una ventana independiente de la ventana principal, en cualquier modo de operación que contiene la información de Log.
2. Limpiar la ventana de Log de simulaciones anteriores por medio de un botón ubicado en cada una de las ventanas correspondientes.
3. Exportar el Log de la Simulación en formato CSV por medio del botón ubicado en la ventana correspondiente en ambos modos de comunicación.
4. Finalizar la aplicación manteniendo la configuración de la última simulación.
5. Devolver la configuración de Simulación a los valores por defecto en cualquier modo de operación a través de un botón específico ubicado en la ventana principal de la aplicación.
6. Representar en una tabla la configuración de los dispositivos Slave activos, mostrando Slave ID y estado en la ventana de Simulación.
7. Mostrar un panel de información general con el modo actual de comunicación, el tipo de comunicación, tipo y versión de Sistema Operativo y el botón de inicio y parada en la ventana de Simulación.
8. Agregar dispositivos a la tabla de dispositivos configurados a través de un panel acoplado de configuración.

5.3 PLANEACIÓN POR CARACTERÍSTICA

5.3.1 Plantilla tercer proceso de la FDD

Criterio de entrada

El equipo de Lista de Características ha completado exitosamente el Proceso 2 de la FDD. Construir la lista de características.

Tareas

Formación Equipo de Planeación	**Director de Proyecto**	**Requerido**
El Equipo de planeación está compuesto por el Director de Proyecto y Programadores Jefe.		

Determinar la Secuencia de Desarrollo	**Equipo de Planeación**	**Requerido**
El Equipo de Planeación asigna una fecha (mes y año únicamente) para terminación de cada conjunto de características. La identificación del conjunto de características y la fecha de terminación (y así, la secuencia de desarrollo) está basada en: • Dependencia entre las características en términos de las clases implicadas. • Balanceo de las cargas a través de los propietarios de las clases.		

• Presentación de los conjuntos de características más complejos y delicados. • Consideración de cualquier evento externo (visible), tales como betas, previews, puntos de retroalimentación y los productos que satisfacen dichos eventos.

Asignar el Feature Set a los Programadores Jefe	**Equipo de Planeación**	**Requerido**
El Equipo de planeación asigna a los Programadores Jefe como propietarios de los conjuntos de características. Esta asignación está basada en: • La secuencia de desarrollo. • Dependencia entre las características en términos de las clases implicadas.		

Asignar clases a los Desarrolladores	**Equipo de Planeación**	**Requerido**
El equipo de planeación asigna desarrolladores como propietarios de clase. Los desarrolladores múltiples clases. La asignación de clases a los desarrolladores está basada en: • Balance de carga de trabajo para los desarrolladores. • Complejidad de las clases. • Uso esperado de las clases. • Secuencia de desarrollo.		

Verificación

Autoevaluación	**Equipo de Planeación**	**Requerido**
La planeación es una actividad de equipo, por lo tanto la autoevaluación es lograda a través de la activa participación del Director del Proyecto y el Programador en Jefe.		

Criterio de Salida

Para la salida del proceso, el equipo de planeación debe producir el plan de desarrollo a satisfacción del Director de Proyecto y el Director de Desarrollo. El plan de desarrollo consiste de:

- Conjunto de características con fechas de terminación (mes y año).
- Conjunto principal de características con fechas de terminación (mes y año) derivadas de la última fecha de terminación de su respectiva lista de características.
- Lista de clases y desarrolladores propietarios (Lista de propietarios de las clases).

5.3.2 Desarrollo tareas plantilla tercer proceso de la FDD

5.3.2.1 Formación del equipo de planeación

Director de Proyecto	**Prof. Fernando Rojas**
Programador en Jefe	Angel Andres Vargas Esteban

5.3.2.2 Secuencia de desarrollo

A continuación se presenta la tabla que muestra la secuencia de desarrollo del proyecto, se registra de manera específica cada una de las características predefinidas en el paso dos de la FDD con las fechas asignadas para su realización, a que Main Feature Set y Feature Set pertenecen y una numeración para una más fácil identificación.

Tabla 6. Secuencia de desarrollo

Conjunto Principal de Características (Main Feature Set)	Conjunto de Características (Feature Set)	Característica (Feature)		Fecha de inicio	Fecha de terminación
MFS 1: Realizar Simulación Master Inicio: Noviembre 25 de 2009 Finalización: Diciembre 18 de 2009	FS 1: Realizar Simulación Master TCP	F01	Seleccionar el modo de operación Master y tiempo de espera máximo de respuesta, para el objeto Device.	25/11/ 2009	25/11/2009
		F02	Seleccionar tipo de comunicación TCP en el objeto Com.	25/11/2009	25/11/2009
		F03	Asignar la dirección IP, y el puerto del dispositivo para configurar ComMode.	26/11/2009	26/11/2009
		F04	Seleccionar un número de registro con tipo RegDesc y almacenarlo en RegSelect	27/11/2009	29/11/2009
		F05	Realizar consulta Modbus con NModbus y almacenar resultados en Query.	30/11/ 2009	30/11/2009
		F06	Definir un intervalo de tiempo y realizar consultas Modbus automáticas separadas por dicho intervalo y almacenar los resultados en Query.	1/12/2009	1/12/2009
		F07	Escribir un registro Modbus con NModbus en un dispositivo esclavo externo y almacenar los resultados en Query.	2/12/ 2009	2/12/2009
		F08	Definir un rango para generar un número aleatorio, generarlo y escribir el número generado en un registro Modbus de un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.	3/12/2009	3/12/2009
		F09	Definir un intervalo de tiempo y realizar escrituras Modbus automáticas de números generados aleatoriamente en un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.	4/12/2009	4/12/2009
		F10	Visualizar respuesta, tipo de respuesta, su representación en HEX y el tiempo de respuesta de la última consulta en la ventana de Simulación en modo Master.	5/12/2009	6/12/2009
		F11	Agregar un renglón con una cadena que contiene la consulta y la respuesta para visualizar la consulta y su resultado en la ventana de Log.	7/12/ 2009	8/12/2009
	FS 2:	F01	Seleccionar el modo de operación Master y tiempo de	9/12/2009	9/12/2009

	Realizar Simulación Master ASCII / Master RTU		espera máximo de respuesta, para el objeto Device.		
		F02	Seleccionar tipo de comunicación RTU o ASCII en el objeto Com.	9/12/ 2009	9/12/2009
		F03	Asignar el puerto serial, baud rate, bits de datos, paridad y bits de parada para configurar ComMode.	9/12/ 2009	9/12/2009
		F04	Seleccionar un número de registro con tipo RegDesc y almacenarlo en RegSelect	10/12/2009	10/12/2009
		F05	Realizar consulta Modbus con NModbus y almacenar resultados en Query.	11/12/2009	11/12/2009
		F06	Definir un intervalo de tiempo y realizar consultas Modbus automáticas separadas por dicho intervalo y almacenar los resultados en Query.	12/12/2009	14/12/2009
		F07	Escribir un registro Modbus con NModbus en un dispositivo esclavo externo y almacenar los resultados en Query.	15/12/2009	15/12/2009
		F08	Definir un rango para generar un número aleatorio, generarlo y escribir el número generado en un registro Modbus de un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.	16/12/2009	16/12/2009
		F09	Definir un intervalo de tiempo y realizar escrituras Modbus automáticas de números generados aleatoriamente en un dispositivo esclavo externo Fusando NModbus y almacenar los resultados en Query.	17/12/2009	17/12/2009
		F10	Visualizar respuesta, tipo de respuesta, su representación en HEX y el tiempo de respuesta de la última consulta en la ventana de Simulación en modo Master.	18/12/ 2009	18/12/2009
		F11	Almacenar la consulta y su resultado en Log.	18/12/ 2009	18/12/2009
MFS 2: Realizar Simulación Slave Inicio: Diciembre 19 de 2009	FS 1: Realizar Simulación Slave TCP	F01	Seleccionar el modo de operación Slave y tiempo de espera máximo de respuesta, para el objeto Com.	19/12/ 2009	19/12/2009
		F02	Seleccionar tipo de comunicación TCP en el objeto Com.	19/12/2009	19/12/2009
		F03	Asignar la dirección IP, y el puerto del dispositivo para configurar ComMode.	19/12/2009	19/12/2009
		F04	Seleccionar un número de registro con tipo RegDesc y almacenarlo en RegSelect	20/12/ 2009	21/12/2009

Finalización: Enero 12 de 2010		F05	Escuchar y responder consultas en un registro o grupo de registros con un valor determinado por el usuario con Listen.	04/01/2010	04/01/2010
		F06	Escuchar y responder consultas en un registro o grupo de registros con un valor generado aleatoriamente dentro de un rango determinado por el usuario.	05/01/ 2010	05/01/2010
		F07	Almacenar la respuesta en Log.	06/01/2010	06/01/2010
	FS 2: Realizar Simulación Slave ASCII/ Slave RTU	F01	Seleccionar el modo de operación Slave y tiempo de espera máximo de respuesta, para el objeto Com.	07/01/2010	07/01/2010
		F02	Seleccionar tipo de comunicación ASCII o RTU en el objeto Com.	07/01/2010	07/01/2010
		F03	Asignar el puerto serial, baud rate, bits de datos, paridad y bits de parada para configurar ComMode.	07/01/2010	07/01/2010
		F04	Seleccionar un número de registro con tipo RegDesc y almacenarlo en RegSelect	08/01/2010	08/01/2009
		F05	Escuchar y responder consultas en un registro o grupo de registros con un valor determinado por el usuario con Listen.	09/01/2010	09/01/2010
		F06	Escuchar y responder consultas en un registro o grupo de registros con un valor generado aleatoriamente dentro de un rango determinado por el usuario.	10/01/2010	11/01/2010
		F07	Almacenar la respuesta en Log.	12/01/2010	12/01/2010
MFS 3: Características GUI Inicio: Enero 13 de 2009 Finalización: Enero 21 de 2009	FS 1: Características GUI	F01	Mostrar una ventana independiente de la ventana principal, en cualquier modo de operación que contiene la información de Log.	13/01/2010	13/01/2010
		F02	Limpiar la ventana de Log de simulaciones anteriores por medio de un botón ubicado en cada una de las ventanas correspondientes.	14/01/2010	14/01/2010
		F03	Exportar el Log de la Simulación en formato CSV por medio del botón ubicado en la ventana correspondiente en ambos modos de comunicación.	15/01/2010	15/01/2010
		F04	Finalizar la aplicación manteniendo la configuración de la última simulación.	16/01/2010	16/01/2010
		F05	Devolver la configuración de Simulación a los valores por	17/01/2010	17/01/2010

			defecto en cualquier modo de operación a través de un botón específico ubicado en la ventana principal de la aplicación.		
		F06	Representar en una tabla la configuración de los dispositivos Slave activos, mostrando Slave ID y estado en la ventana de Simulación.	19/01/2010	19/01/2010
		F07	Mostrar un panel de información general con el modo actual de comunicación, el tipo de comunicación, tipo y versión de Sistema Operativo y el botón de inicio y parada en la ventana de Simulación.	20/01/2010	20/01/2010
		F08	Agregar dispositivos a la tabla de dispositivos configurados a través de un panel acoplado de configuración.	21/01/ 2010	21/01/2010

Fuente: *Autores del proyecto*

5.3.2.3 Asignación de los Feature Sets a los programadores jefe

En el caso particular de este proyecto debe aclararse que debido a su naturaleza, y a su reducido número de integrantes, el Programador en Jefe es el propietario único que todas las clases involucradas en el modelo, por lo tanto las clases son únicamente divididas teniendo en cuenta los parámetros de la FDD para los dos desarrolladores de este proyecto.

5.3.2.4 Asignación de clases a los desarrolladores

Para esta última tarea, la cual consiste en la asignación de clases a los desarrolladores, los integrantes del equipo de desarrollo, decidieron realizar un cambio a los lineamientos de la FDD, en este aspecto se consideró la posibilidad de que en la etapa de diseño el modelo de operación tuviera cambios importantes, por lo tanto la asignación de clases fue una tarea trasladada al cuarto proceso de la FDD donde tanto todas las clases y métodos están perfectamente identificados y es posible una clara y eficaz asignación a los desarrolladores.

Estos cambios importantes que pueden llegar a presentarse en el modelo de operación después de las primeras tareas del proceso cuatro de la FDD no son ajenos a la metodología que claramente manifiesta que ésto puede suceder durante el desarrollo de cualquier proyecto de software, ya que el modelo de operación obtenido en el proceso uno es más una abstracción de la situación a tratar que un modelo de implementación.

El modelo de operación obtenido en la primera etapa de la FDD sirve como base o inspiración para la realización del modelo de implementación, dentro de este proceso pueden surgir nuevas clases, clases ya existentes puede convertirse en métodos o viceversa y por supuesto pueden desaparecer algunas de las mismas.

5.4 DISEÑO Y CONSTRUCCIÓN POR CARACTERÍSTICA

Una de las bases de la construcción de software a través de la FDD es el diseño y la construcción de características de manera iterativa. El programador en jefe selecciona una característica para desarrollar, identifica las clases involucradas y asigna propietarios de las clases formando el equipo de característica para esta iteración; también produce el diagrama de secuencia y escribe los encabezados de clases y métodos. Después de una inspección, los propietarios de las clases añaden el código respectivo a sus clases, hacen pruebas unitarias, integran e inspeccionan el código. Una vez el programador en jefe está satisfecho, la característica completa es promovida a la construcción principal, y los procesos 4 y 5 se repiten con los siguientes grupos e características.

Este proceso iterativo será ilustrado en la sección 5.4.2 a través del desarrollo de una de las características formuladas en el primer proceso de la FDD.

5.4.1 Aspectos relevantes sobre el desarrollo de Simbus

Desarrollar una aplicación en equipo, multiplataforma y que cumpla con los estándares del código abierto, requiere de herramientas especiales para la consecución de este objetivo. En esta sección se dan a conocer estas herramientas especiales y las implicaciones que trajo su uso sobre este proyecto.

5.4.1.1 Subversion SVN

Subversion SVN es un sistema de control de versiones de código abierto. Esto es, Subversion maneja archivos y directorios en el tiempo. Un árbol de archivos es colocado en un repositorio central. El repositorio es más como un servidor de archivos, excepto que este recuerda cada cambio alguna vez hecho en los archivos y directorios, esto permite recuperar antiguas versiones de los datos, o examinar la historia de cómo los datos han cambiado. Considerando esto, muchas personas piensan en un sistema de control de versiones como en un tipo de "máquina de tiempo".

Subversion puede ingresar al repositorio a través de redes, las cuales permiten ser usadas por muchas personas en diferentes computadores. A cierto nivel, la habilidad para varias personas de modificar y administrar el mismo conjunto de datos desde su respectiva localización promueve la colaboración y el progreso puede ocurrir mucho más rápido. Debido a que el trabajo es versionado, no hay que temer que se sacrifique calidad por la pérdida de linealidad, si algún cambió incorrecto es hecho a los datos, tan solo se requiere deshacer el cambio y se estará así de nuevo en la versión anterior libre de errores.

Esto permitió al equipo de desarrolladores trabajar de forma paralela en diferentes áreas del desarrollo, además de llevar un control detallado de cualquier cambio realizado en la documentación o código fuente.

5.4.1.2 Librerías GTK+ y Gtk#

GTK+ es un kit de herramientas altamente usable para la creación gráfica de interfaces de usuario la cual tiene compatibilidad entre diferentes plataformas y una interfaz de programaciones de aplicaciones API fácil de usar. GTK+ está escrita en C, pero tiene bindings[11] para muchos otros lenguajes de programación populares tales como C++, Python y C#. GTK+ esta bajo la licencia GNU LGPL

[11] "Envoltorios" o "cubiertas" que permiten que GTK+ sea usada desde otros lenguajes de programación.

2.1[12] permitiendo el desarrollo tanto de software tanto abierto como propietario sin cobros o legalidades.

Gtk# es un kit de herramientas para realizar interfaces gráficas de usuario especial para Mono y .Net. El proyecto une el kit de herramientas GTK+ y el surtido de librerías de GNOME, permitiendo el desarrollo de aplicaciones Gnome gráficas completamente nativas usando los entornos de desarrollo Mono y .Net.

Algunos de los principales inconvenientes en el proyecto se debieron al uso de las librerías GTK+, principalmente a la falta de documentación de las librerías Gtk# y a la falta de componentes de uso especifico de la librería como menús emergentes o contextuales y grillas para visualizar información, en general debidos la complejidad en el manejo de los componentes GTK.

5.4.2 Plantillas cuarto y quinto proceso de la FDD

5.4.2.1 Plantilla cuarto proceso de la FDD

FDD Proceso 4: Diseñar por característica

Criterio de entrada

El equipo de planeación ha completado exitosamente le proceso 3 de la FDD. Planear por característica.

Tareas

Conducir una guía del dominio	**Experto en el dominio**	**Requerido**
El equipo de característica desarrolla diagramas de secuencia detallados requeridos para cada característica que está siendo diseñada. El equipo escribe y almacena cualquier diseño alternativo, decisiones de diseño, clarificación de requerimientos y notas alternativas en el diseño o notas sobre el diseño del paquete.		

Refinar el Modelo de Operación	**Programador en Jefe**	**Requerido**
El Programador en Jefe refina el modelo, añadiendo clases adicionales, operaciones y/o atributos, o haciendo cambios a las clases existentes basado en los diagramas de secuencia definidos por las características.		

[12] GNU Lesser General Public License version 2.1

Tarea trasladada por el equipo de desarrollo desde el proceso tres

Asignar clases a los Desarrolladores	Equipo de Planeación	Requerido
El equipo de planeación asigna desarrolladores como propietarios de clase. Los desarrolladores múltiples clases. La asignación de clases a los desarrolladores está basada en: • Balance de carga de trabajo para los desarrolladores. • Complejidad de las clases. • Uso esperado de las clases. • Secuencia de desarrollo.		

Formación Equipo de Características	Programador en Jefe	Requerido
El programador en jefe identifica las clases que probablemente estarán involucradas en el diseño de un determinado grupo de características. A partir de lista de propietarios de las clases, el Programador en Jefe identifica los desarrolladores necesarios para formar el equipo de característica. Como parte de este paso, el Programador en Jefe inicia el diseño de un nuevo paquete como parte del paquete de trabajo.		

Escribir los encabezados de las clases y métodos	Equipo de Característica	Requerido
Usando los archivos fuente en el lenguaje de implementación actualizado del "Modelo de Operación Refinado", cada propietario de clase escribe los encabezados de los métodos y las clases para cada uno de los objetos definidos en la característica y en los diagramas de secuencia. Esto incluye tipos de parámetros, tipos retornados, excepciones y mensajes.		

Verificación

Inspección del Diseño	Equipo de Característica	Requerido
Un diseño de inspección exitoso es la verificación de la salida de este proceso. La tarea de inspección del diseño es descrita a continuación, como el criterio de salida del proceso 4 de la FDD.		

Criterio de Salida

Para finalizar este proceso, el equipo de característica debe producir un paquete diseñado, inspeccionado y exitoso. El paquete diseñado comprende.

- Un documento que integra y describe el paquete diseñado.
- Los diagramas de secuencia.

- Diseños alternativos (si existen)
- El modelo del objeto con las clases, métodos y atributos nuevos o actualizados.
- Los encabezados de las clases, métodos o atributos realizados en la herramienta de desarrollo seleccionada para el proyecto.

5.4.2.2 Plantilla quinto proceso de la FDD

FDD Proceso 5: Construir por característica

Criterio de entrada

El equipo de característica ha finalizado exitosamente para las características seleccionadas el proceso 4 de la FDD, Diseñar por Característica. Esto significa, el diseño del paquete del proceso 4 ha sido exitosamente inspeccionado.

Tareas

Implementar Clases y Métodos	**Equipo de Característica**	**Requerido**
Los propietarios de las clases implementan los elementos necesarios para satisfacer los requerimientos de sus clases para las características respectivas en el paquete de trabajo.		

Conducir una Inspección de Código	**Equipo de Característica**	**Requerido**
El equipo de característica conduce una inspección del código, tanto antes como después de la tarea de Prueba Unitaria.		

Prueba Unitaria	**Equipo de Característica**	**Requerido**
Cada propietario de clase prueba su código para asegurar que todos los requerimientos que involucren sus clases se satisfagan.		

Promoción a Construcción	**Programador en Jefe, Equipo de Característica**	**Requerido**
Las clases pueden promoverse para la construcción solo después de una inspección de código y pruebas unitarias exitosas, el Programador en Jefe es el punto de integración para el total de las características y el responsable de dar promoción a las clases incluidas.		

Verificación

Inspección de Código y Prueba Unitaria	Programador en Jefe, Equipo de Característica	Requerido
Una exitosa inspección de código más una terminación satisfactoria de la prueba unitaria es la verificación de este proceso.		

Criterio de Salida

Para salir de este proceso, el equipo de característica debe completar el desarrollo de las funciones significativas para el cliente.

5.4.3 Iteración característica seleccionada

La característica seleccionada para mostrar una de las iteraciones de proceso de desarrollo de Simbus es:

- Asignar la dirección IP, y el puerto del dispositivo para configurar ComMode.

Perteneciente al Main Feature Set 2 y Feature Set 1.

Es importante aclarar que las tareas relacionadas con el diseño (Proceso cuatro FDD), fueron realizadas por ambos desarrolladores.

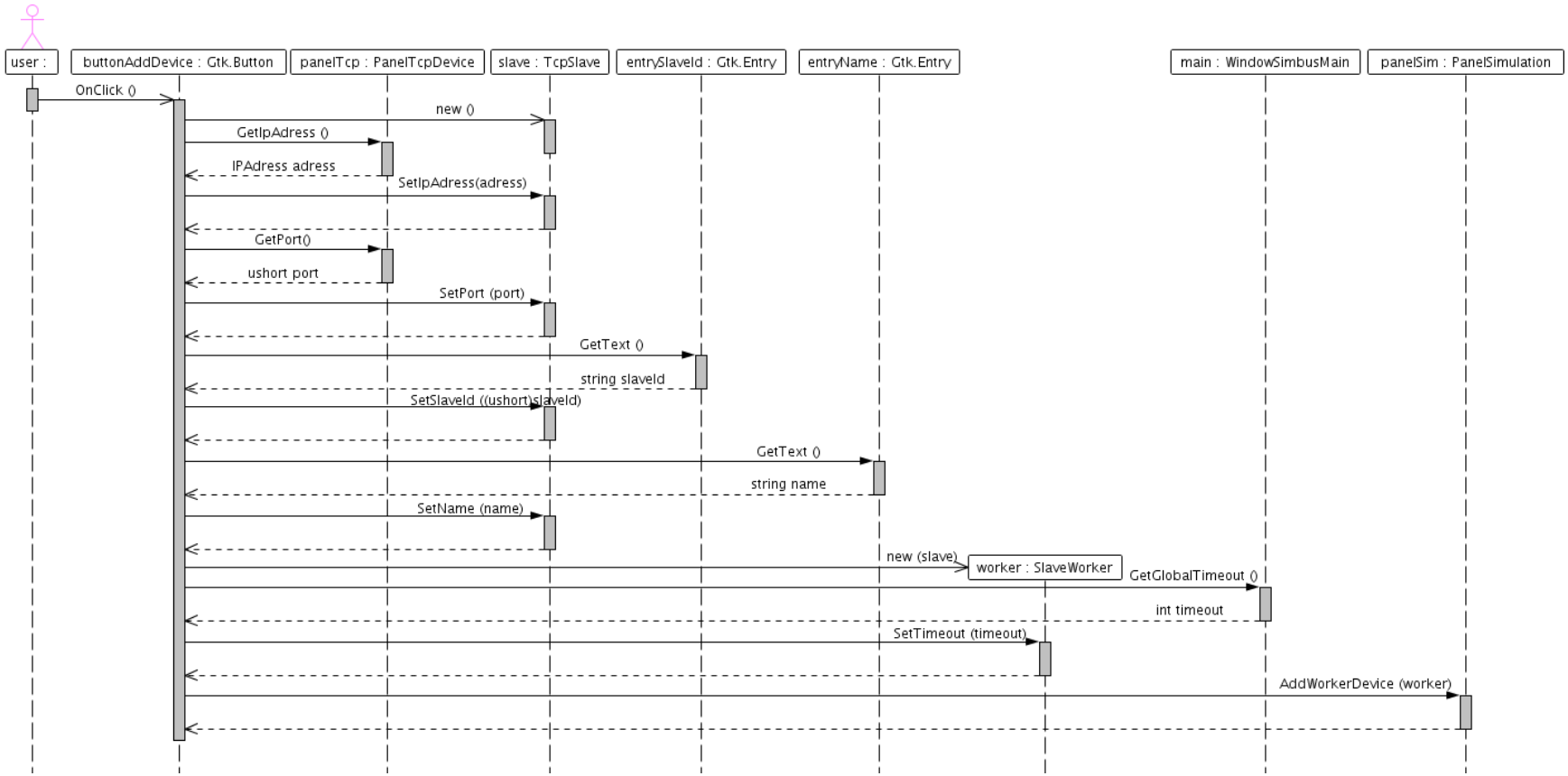
El proceso iterativo que se llevará a cabo puede ser observado con más detalle en las revisiones 135, 136, 137 y 138 del sistema de control de versiones Subversion SVN, desde la inclusión del diagrama de secuencia hasta la generación del tag con la imagen del árbol y la nueva característica implementada.

5.4.3.1 Conducción guía del dominio

En el desarrollo de esta tarea y con la activa participación del programador en jefe, se pormenoriza a través del diagrama de secuencia la forma de implementación de la característica seleccionada. De esta manera el equipo de desarrollo obtiene un entendimiento minucioso y exacto de los objetos, métodos y algoritmos a utilizar en la implementación.

El diagrama de secuencia añadido en la documentación UML durante esta etapa se muestra en la figura 28, en algunos casos debido a una posible intrínseca relación de la característica con otras, estas pueden compartir el mismo diagrama de secuencia.

Figura 28. Diagrama de secuencia para la característica seleccionada



Fuente: *Autores del proyecto.*

5.4.3.2 Refinación del modelo de operación

Cada una de las iteraciones puede producir un cambio o mejoría en el modelo, es decir, va siendo refinado, convirtiéndose finalmente en el modelo de implementación. También se producen cambios en el modelo de la GUI, que fue realizada de forma totalmente independiente del modelo de implementación, siguiendo los lineamientos del Modelo Vista Controlador o MVC[13].

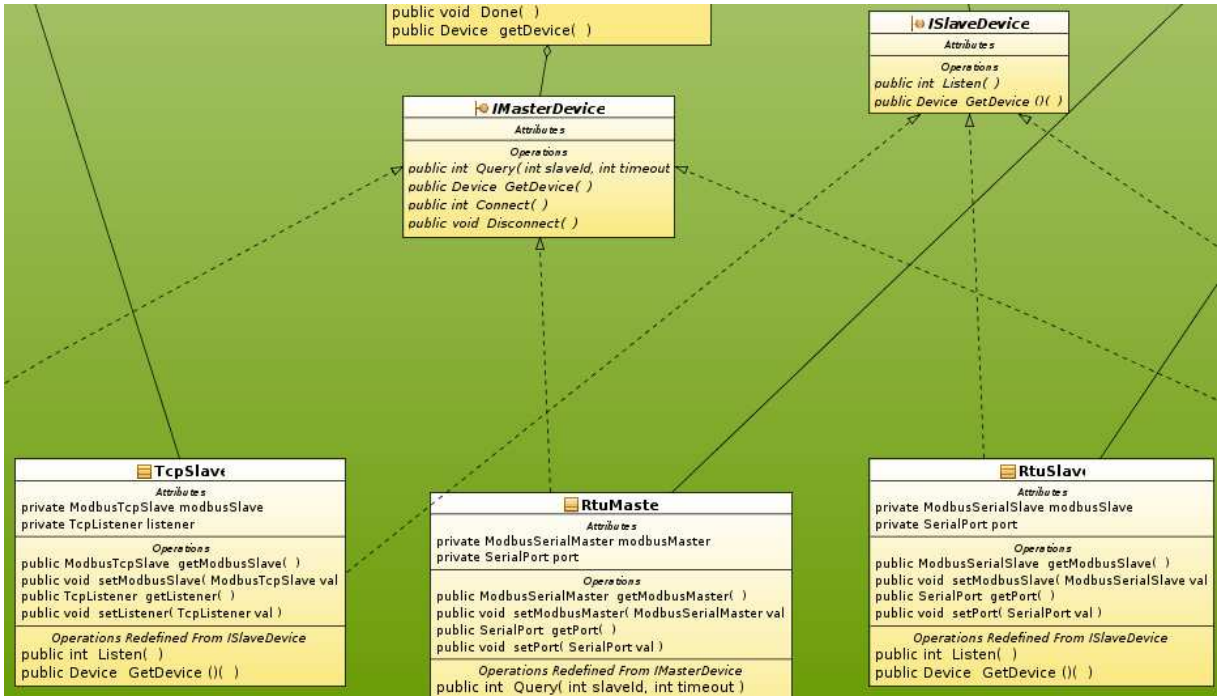
El Modelo Vista controlador es un tipo de arquitectura de software que separa los datos de la aplicación de la vista y de la lógica de control, en el caso de Simbus[14] el *modelo* que es la representación específica de la información con la cual el sistema opera está expresado por completo en el modelo de implementación llamado ModCore. La *vista o interfaz de usuario* que presenta el modelo en un formato adecuado para interactuar, está dada por el modelo GUI.

Finalmente el *controlador* es el que responde a eventos, usualmente acciones del usuario, e invoca cambios en el modelo y, probablemente, en la vista. A continuación se muestra el detalle del componente ModCore donde se añadieron los métodos de la interfaz ISlaveDevice, Figura 29.

[13] Patrón descrito por primera vez en 1979 por *Trygve Reenskaug*. La implementación original está descrita a fondo en *Programación de Aplicaciones en Smalltalk-80(TM): Como utilizar Modelo Vista Controlador.*

[14] Nombre dado por los desarrolladores de este proyecto a la aplicación realizada.

Figura 29. Detalle componente Modcore



Fuente: *Autores del proyecto*

Los diagramas completos del componente ModCore y el componente GUI o de interfaz gráfica se pueden observar en las figuras 38 y 39 y también en el anexo digital de este documento.

5.4.3.3 Asignación de clases a los desarrolladores

Identificando las clases asociadas se asignan a uno de los desarrolladores.

Tabla 7. Asignación de clases

Clases Asociadas	Desarollador Asignado
Clases componente ModCore	
TcpSlave	Angel Vargas
SlaveDevice	
SlaveWorker	
Clases componente GUI	
PanelDevMode	Angel Vargas
PanelSimulation	

Fuente: *Autores del proyecto.*

5.4.3.4 Escritura de encabezados

La interfaz ISlaveDevice que había sido contemplada en el diseño de implementación pero de la cual solo estaba disponible la cabecera de su declaración, se modificó agregando los esqueletos de los métodos que se quieren en cada clase que hereda de ella, Figura 30.

Figura 30. Esqueleto métodos interfaz ISlaveDevice

```
using System;

namespace Simbus.ModCore
{
    ///
    public interface ISlaveDevice   {

        ///
        int Listen();

        ///
        Device GetDevice ();
    }
}
```

Fuente: *Autores del proyecto.*

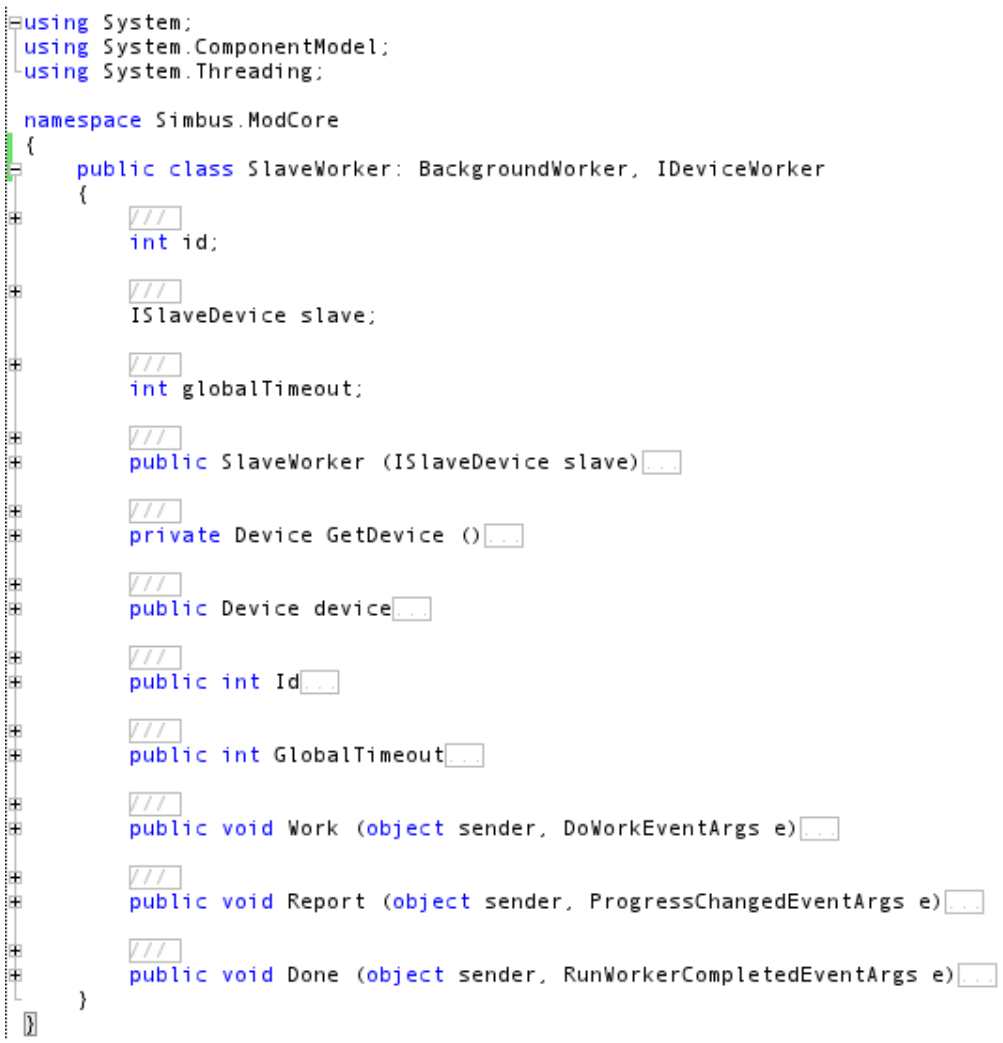
Siendo esta la primera característica que requería la implementación de la clase SlaveWroker y además siendo esta la única clase requerida por esta característica que no se encontraba implementada, es el único encabezado de clase agregado en esta iteración. El código respectivo al encabezado se muestra en la figura 31.

5.4.3.5 Implementar clases y métodos

Se incluyen porciones del código fuente para ilustrar a implementación de la característica; no todo el código ha sido incluido por razones prácticas, ver figura 32.

Los métodos parciamente implementados en la clase TcpSlave, ver figura 33, aunque no sean parte explicita de esta característica deben definirse prematuramente al heredar de la interfaz ISlaveDevice, ya que el componente no se construiría.

Figura 31. Esqueleto clase SlaveWorker



```
using System;
using System.ComponentModel;
using System.Threading;

namespace Simbus.ModCore
{
    public class SlaveWorker: BackgroundWorker, IDeviceWorker
    {
        ///
        int id;

        ///
        ISlaveDevice slave;

        ///
        int globalTimeout;

        ///
        public SlaveWorker (ISlaveDevice slave)...

        ///
        private Device GetDevice ()...

        ///
        public Device device...

        ///
        public int Id...

        ///
        public int GlobalTimeout...

        ///
        public void Work (object sender, DoWorkEventArgs e)...

        ///
        public void Report (object sender, ProgressChangedEventArgs e)...

        ///
        public void Done (object sender, RunWorkerCompletedEventArgs e)...
    }
}
```

Fuente: *Autores del proyecto.*

Figura 32. Clase SlaveWorker

```
        public int Id {
            get {return id; }
            set {id = value; }
        }

        ///
        public int GlobalTimeout {
            get {return globalTimeout; }
            set {globalTimeout = value; }
        }

        /// <summary>
        /// Method that handles the <see cref="SlaveWorker"/> <see cref="DoWork"/> event.
        /// </summary>
        /// <param name="sender">
        /// A <see cref="System.Object"/>
        /// </param>
        /// <param name="e">
        /// A <see cref="DoWorkEventArgs"/>
        /// </param>
        public void Work (object sender, DoWorkEventArgs e)
        {
            // First we try to establish a link with the slave.
            if (!(slave.Listen () == 0)) {
                Console.WriteLine ("Error al iniciar la escucha");
                return;
            }
            while (!CancellationPending) {
                Thread.Sleep (1000);
            }
            // Disconnect from the slave.
            //slave.Disconnect ();
        }
```

Fuente: *Autores del proyecto.*

Figura 33. Clase TcpSlave

```
        /// <summary>
        /// Listens any master query.
        /// </summary>
        /// <returns>
        /// A <see cref="System.Int32"/>
        /// </returns>
        public int Listen ()
        {
            //Add some code here XD
            OnDeviceStateChange (this, "");
            return 0;
        }

        /// <summary>
        /// Returns the current instance.
        /// </summary>
        /// <returns>
        /// A <see cref="Device"/> representation of the current <see cref="TcpSlave"/>.
        /// </returns>
        public Device GetDevice ()
        {
            return this;
        }
```

Fuente: *Autores del proyecto.*

Dada la versatilidad de a implementación del componente GUI para completar la característica solo se requiere agregar el caso específico para un dispositivo TCP Slave.

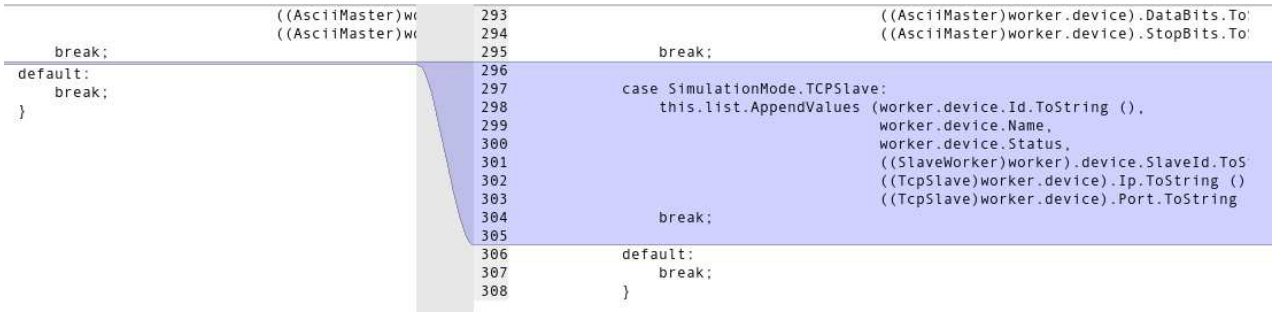
Figura 34. Caso agregado en la clase PanelDevMode

```
    device = new TcpMaster ();
    break;
case SimulationMode.RTUMaster:
    ((RtuMaster)device).Id = 0;
    ((RtuMaster)device).PortName = panelSerialDevice.GetPortName ();
    main.panelSim.AddWorkerDevice (masterAsciiWorker);
    device = new AsciiMaster ();
    break;
default:
    break;
}
```

```
case SimulationMode.TCPSlave:
    ((TcpSlave)device).Ip = panelTcpDevice.GetIpAdress ();
    ((TcpSlave)device).Ip = panelTcpDevice.GetIpAdress ();
    ((TcpSlave)device).Port = panelTcpDevice.GetPort ();
    ((TcpSlave)device).SlaveId = Convert.ToByte(spinbuttonSlaveId.Text
    ((TcpSlave)device).Name = (entryDeviceName.Text.Trim ());

    // Creating the workerDevice:
    SlaveWorker slaveWorker = new SlaveWorker ((TcpSlave)device);
    slaveWorker.GlobalTimeout = main.GlobalTimeout;

    main.panelSim.AddWorkerDevice (slaveWorker);

    device = new TcpSlave ();
    break;

default:
    break;
}
```

Fuente: *Autores del proyecto.*

Figura 35. Caso agregado en la clase PanelSimulation

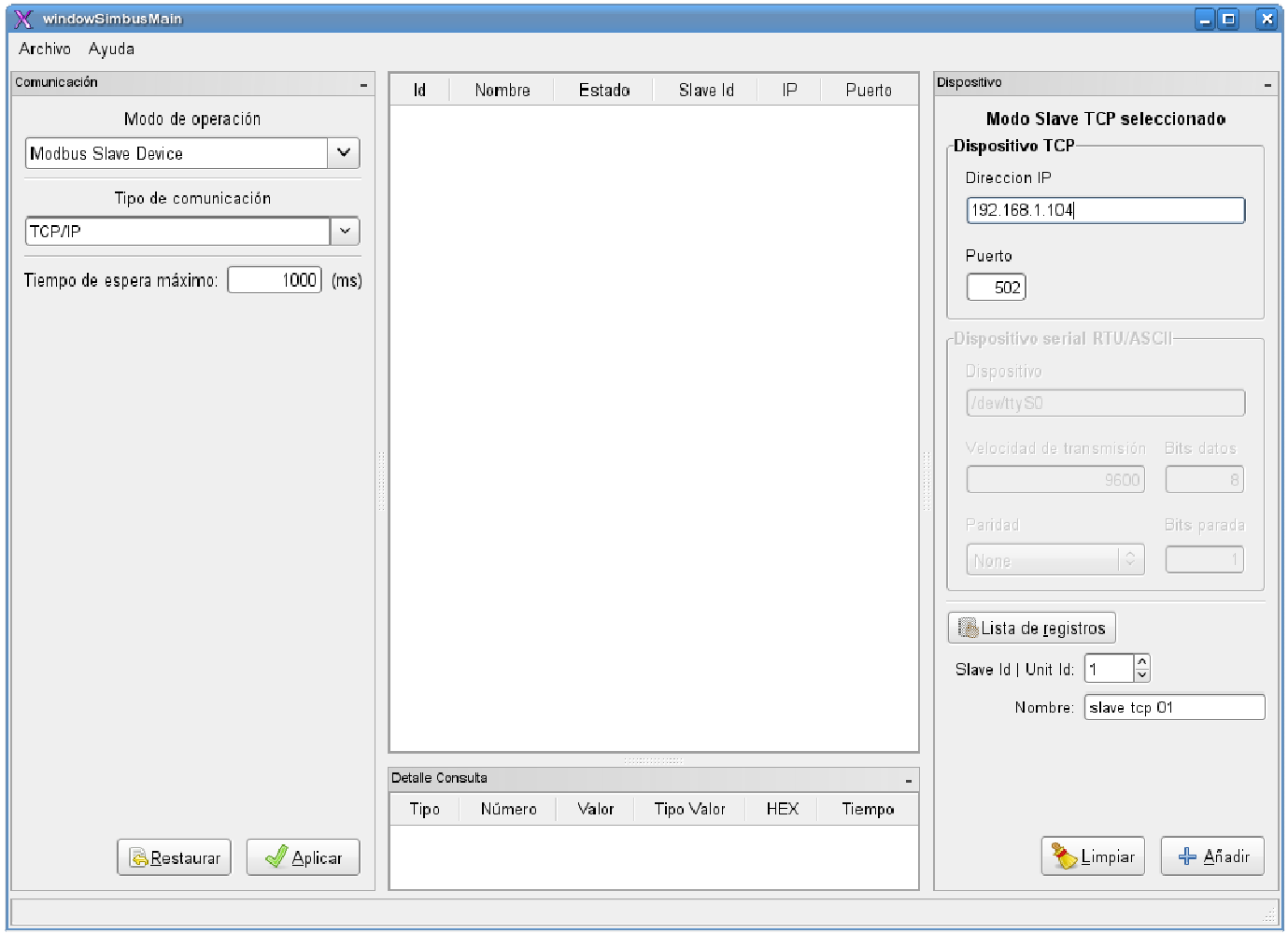


Fuente: *Autores del proyecto.*

5.4.3.6 Prueba unitaria

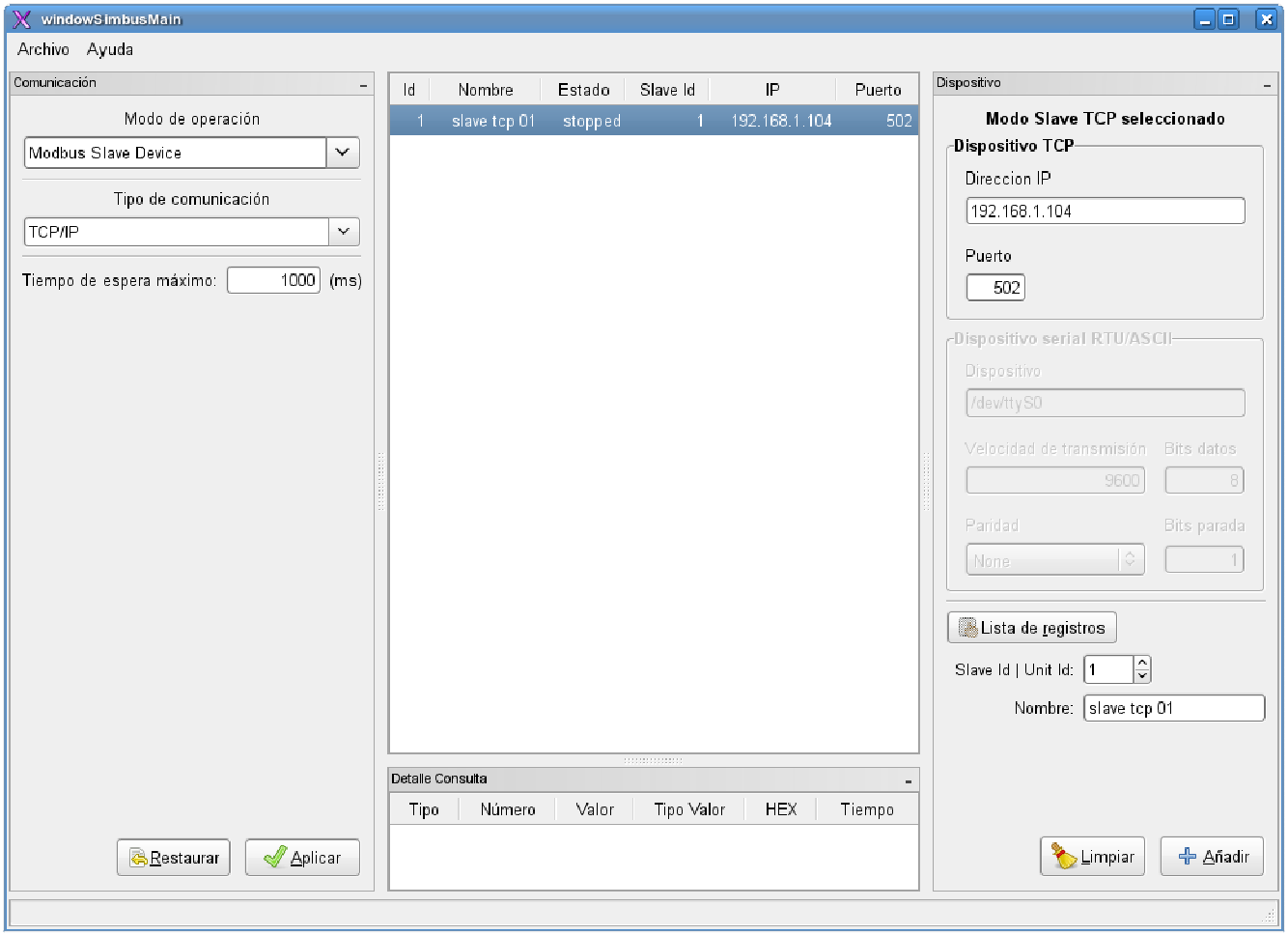
Después de la construcción de cada uno de los componentes de los cuales depende la GUI se ejecuta la aplicación principal y se prueba la nueva característica, en las siguientes imágenes se puede observar la verificación del correcto funcionamiento de dicha característica.

Figura 36. Simbus GUI, ingreso información de IP y puerto



Fuente: *Autores del proyecto.*

Figura 37. Simbus GUI, dispositivo agregado a la lista de dispositivos



Fuente: *Autores del proyecto.*

5.4.3.7 Promoción a construcción

Como en este caso la característica se desarrolla con todos los componentes propuestos en el diseño el proceso de promoción a construcción es omitido, en cambio haciendo uso del sistema de control de versiones SVN lo que se conoce como “tag”, que no es más que una instantánea del árbol del proyecto justo en el estado después de incluir la característica, esto con la finalidad de poder llevar un histórico de los cambios realizados y si es necesario volver a una versión anterior del proyecto.

6. RESULTADOS FINALES

6.1 CARACTERÍSTICAS FINALES SIMBUS

Después de la realización de todas las iteraciones necesarias para cumplir con cada una de las características determinadas, se obtienen los resultados planteados para este proyecto.

En el capítulo 4 se describió el proceso de portar la librería NModbus 1.8.0 al Framework Mono 2. NModbus se encuentra en una versión compilada para el Framework 2.0 de Microsoft, la migración permite tener una versión compilada de la misma para el Framework multiplataforma Mono versión 2.

A partir de de este punto, donde ya se cuenta con la librería NModbus compilada para el Framework Mono 2, se inició el proceso de diseño y desarrollo de la aplicación, mediante los procesos ya detallados de la FDD, y la documentación de los mismos con UML. En la figura 38 se muestra el Diagrama de clases final después del proceso iterativo y en la figura 39 el Diagrama de clases de la interfaz gráfica de usuario de Simbus. Una vista más detallada de estos diagramas se encuentra en el anexo digital de este documento.

La documentación UML de Simbus incluye también los diagramas de secuencia para cada una de las características planteadas en la fase de diseño, estos se relacionan con cada una de las características a la que pertenecen en la tabla 8 y pueden observarse con más detalle dentro del anexo digital de este documento. También en la tabla se relacionan las clases asociadas a cada característica.

Es importante anotar que la aplicación final junto con el código fuente y la documentación, la cual incluye las imágenes de los diagramas de secuencia, de clases y los documentos fuente también están disponibles en la página principal del proyecto Simbus alojada en SourceForge.net[15].

[15] Página principal proyecto Simbus: *http://sourceforge.net/projects/simbus/*

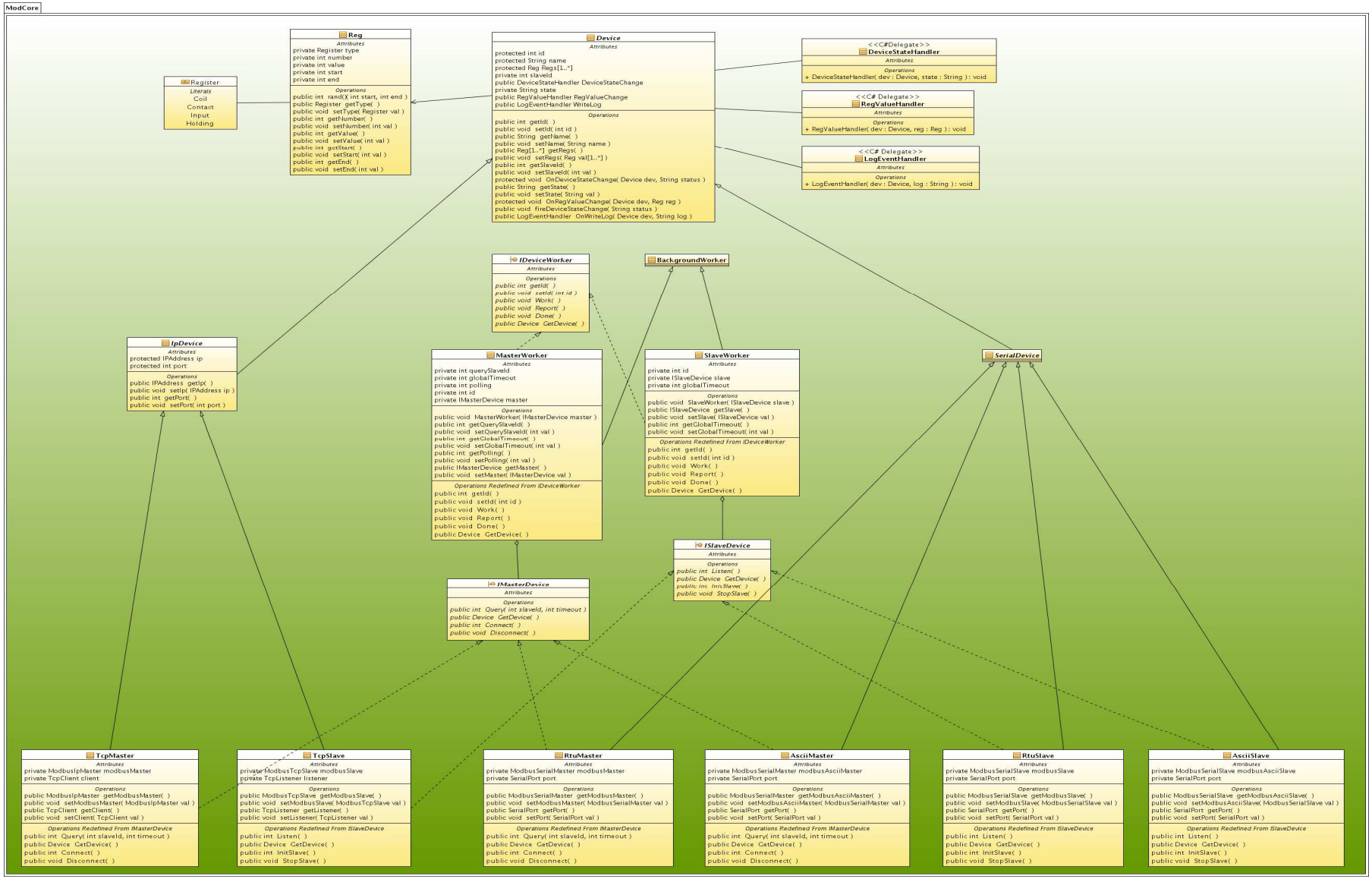


Figura 38. Diagrama de clases Simbus
Fuente: *Autores del proyecto.*

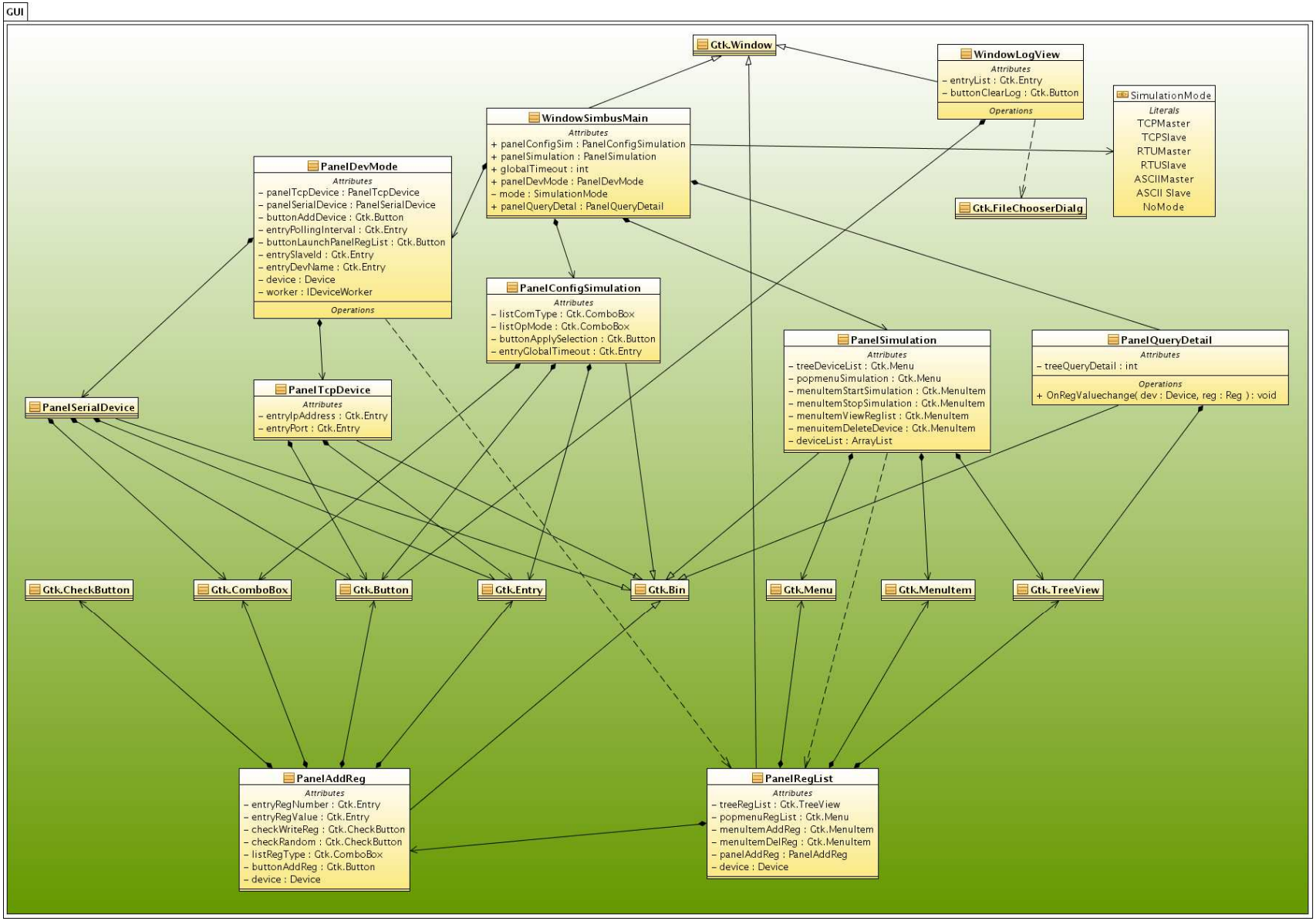


Figura 39. Diagrama de clases interfaz gráfica Simbus
Fuente: *Autores del proyecto.*

Tabla 8. Relación característica y resultados

MFS	FS	Característica (Feature)		Diagrama de secuencia	Clases involucradas componente ModCore	Clases involucradas componente GUI
MFS 1: Realizar Simulación Master	FS 1: Realizar Simulación Master TCP	F01	Seleccionar el modo de operación Master y tiempo de espera máximo de respuesta, para el objeto Device.	MFS01_FS01_F01_F02		PanelSimulation WindowSimbusMain PanelConfigSim
		F02	Seleccionar tipo de comunicación TCP en el objeto Com.	MFS01_FS01_F01_F02		PanelSimulation WindowSimbusMain PanelConfigSim
		F03	Asignar la dirección IP, y el puerto del dispositivo para configurar ComMode.	MFS01_FS01_F03	TcpMaster MasteWorker	PanelDevMode WindowSimbusMain PanelSimulation PanelTcpDevice
		F04	Seleccionar un número de registro con tipo RegDesc y almacenarlo en RegSelect	MFS01_FS01_F04	Device Reg Register	PanelRegList PanelAddReg PanelDevMode
		F05	Realizar consulta Modbus con NModbus y almacenar resultados en Query.	MFS01_FS01_F05	TcpMaster IMasterDevice	PanelSimulation
		F06	Definir un intervalo de tiempo y realizar consultas Modbus automáticas separadas por dicho intervalo y almacenar los resultados en Query.	MFS01_FS01_F06	TcpMaster MasterWorker IMasterDevice	PanelSimulation
		F07	Escribir un registro Modbus con NModbus en un dispositivo esclavo externo y almacenar los resultados en Query.	MFS01_FS01_F07_F08_F09	MasterWorker IMasterDevice TcpMaster Reg Register	
		F08	Definir un rango para generar un número aleatorio, generarlo y escribir el número generado en un registro Modbus de un dispositivo esclavo externo usando NModbus y almacenar	MFS01_FS01_F07_F08_F09	MasterWorker IMasterDevice TcpMaster Reg Register	

			los resultados en Query.			
		F09	Definir un intervalo de tiempo y realizar escrituras Modbus automáticas de números generados aleatoriamente en un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.	MFS01_FS01_F07_F08_F09	MasterWorker IMasterDeovice TcpMaster Reg Register	
		F10	Visualizar respuesta, tipo de respuesta, su representación en HEX y el tiempo de respuesta de la última consulta en la ventana de Simulación en modo Master.	MFS01_FS01_F10	Device MasterWorker IMasterDevice	WindowSimbusMain PanelQueryDetail
		F11	Agregar un renglón con una cadena que contiene la consulta y la respuesta para visualizar la consulta y su resultado en la ventana de Log.	MFS01_FS01_F11	Device MasterWorker IMasterDevice	PanelSim WindowLogView
	FS 2: Realizar Simulación Master ASCII / Master RTU	F01	Seleccionar el modo de operación Master y tiempo de espera máximo de respuesta, para el objeto Device.	MFS01_FS01_F01_F02		PanelSimulation WindowSimbusMain PanelConfigSim
		F02	Seleccionar tipo de comunicación RTU o ASCII en el objeto Com.	MFS01_FS01_F01_F02		PanelSimulation WindowSimbusMain PanelConfigSim
		F03	Asignar el puerto serial, baud rate, bits de datos, paridad y bits de parada para configurar ComMode.	MFS01_FS02_F03	RtuMaster AsciiMaster MasteWorker	PanelDevMode WindowSimbusMain PanelSim PanelSerialDevice
		F04	Seleccionar un número de registro con tipo RegDesc y almacenarlo en RegSelect	MFS01_FS02_F04	Device Reg Register	PanelDevMode PanelRegList PanelAddReg
		F05	Realizar consulta Modbus con NModbus y almacenar resultados en Query.	MFS01_FS01_F05	RtuMaster AsciiMaster IMasterDevice	PanelSimulation
		F06	Definir un intervalo de tiempo y	MFS01_FS01_F06	RtuMaster	

			realizar consultas Modbus automáticas separadas por dicho intervalo y almacenar los resultados en Query.		AsciiMaster MasterWorker IMasterDevice	PanelSimulation
		F07	Escribir un registro Modbus con NModbus en un dispositivo esclavo externo y almacenar los resultados en Query.	MFS01_FS01_F07_F08_F09	MasterWorker IMasterDevice AsciiMaster RtuMaster Reg Register	
		F08	Definir un rango para generar un número aleatorio, generarlo y escribir el número generado en un registro Modbus de un dispositivo esclavo externo usando NModbus y almacenar los resultados en Query.	MFS01_FS01_F07_F08_F09	MasterWorker IMasterDevice AsciiMaster RtuMaster Reg Register	
		F09	Definir un intervalo de tiempo y realizar escrituras Modbus automáticas de números generados aleatoriamente en un dispositivo esclavo externo Fusando NModbus y almacenar los resultados en Query.	MFS01_FS01_F07_F08_F09	MasterWorker IMasterDevice AsciiMaster RtuMaster Reg Register	
		F10	Visualizar respuesta, tipo de respuesta, su representación en HEX y el tiempo de respuesta de la última consulta en la ventana de Simulación en modo Master.	MFS01_FS01_F10	Device MasterWorker IMasterDwevice	WindowSimbusMain PanelQueryDetail
		F11	Almacenar la consulta y su resultado en Log.	MFS01_FS01_F11	Device MasterWorker IMasterDevice	PanelSim WindowLogView
MFS 2: Realizar Simulación Slave	FS 1: Realizar Simulación Slave TCP	F01	Seleccionar el modo de operación Slave y tiempo de espera máximo de respuesta, para el objeto Com.	MFS02_FS01_F01_F02		PanelSimulation WindowSimbusMain PanelConfigSim
		F02	Seleccionar tipo de comunicación TCP en el objeto Com.	MFS02_FS01_F01_F02		PanelSimulation WindowSimbusMain PanelDevMode
		F03	Asignar la dirección IP, y el puerto del dispositivo para configurar ComMode.	MFS02_FS01_F03	TcpSlave SlaveWorker	PanelDevMode WindowSimbusMain PanelSim

						PanelTcpDevice
		F04	Seleccionar un número de registro con tipo RegDesc y almacenarlo en RegSelect	MFS02_FS01_F04	Device Reg Register	PanelDevMode PanelRegList PanelAddReg
		F05	Escuchar y responder consultas en un registro o grupo de registros con un valor determinado por el usuario con Listen.	MFS02_FS01_F05	TcpSlave SlaverWorker ISlaveDevice SlaveDevice	PanelSimulation
		F06	Escuchar y responder consultas en un registro o grupo de registros con un valor generado aleatoriamente dentro de un rango determinado por el usuario.	MFS02_FS01_F06	TcpSlave ISlaveDevice Reg SlaveWorker	PanelSimulation
		F07	Almacenar la respuesta en Log.	MFS02_FS01_F07	TcpSlave SlaveDevice WindowLogView	PanelSimulation
	FS 2: Realizar Simulación Slave ASCII/ Slave RTU	F01	Seleccionar el modo de operación Slave y tiempo de espera máximo de respuesta, para el objeto Com.	MFS02_FS02_F01_F02		PanelSimulation WindowSimbusMain PanelConfigSimulation
		F02	Seleccionar tipo de comunicación ASCII o RTU en el objeto Com.	MFS02_FS02_F01_F02		PanelSimulation WindowSimbusMain PanelConfigSimulation
		F03	Asignar el puerto serial, baud rate, bits de datos, paridad y bits de parada para configurar ComMode.	MFS02_FS02_F03	RtuSlave AsciiSlave SlaveWorker	PanelDevMode WindowSimbusMain PanelSimulation PanelSerialDevice
		F04	Seleccionar un número de registro con tipo RegDesc y almacenarlo en RegSelect	MFS02_FS02_F04	Device Reg Register	PanelDevMode PanelRegList PanelAddReg
		F05	Escuchar y responder consultas en un registro o grupo de registros con un valor determinado por el usuario con Listen.	MFS02_FS02_F05	RtuSlave AsciiSlave SlaveWorker ISlaveDevice SlaveDevice	PanelSimulation
		F06	Escuchar y responder consultas en un registro o grupo de registros con un valor generado	MFS02_FS02_F06	RtuSlave AsciiSlave ISlaveDevice	PanelSimulation

			aleatoriamente dentro de un rango determinado por el usuario.		Reg Register SlaveWorker	
		F07	Almacenar la respuesta en Log.	MFS02_FS02_F07	RtuSlave AsciiSlave SlaveDevice WindowLogView	PanelSimulation
MFS 3: GUI	FS 1: GUI	F01	Mostrar una ventana independiente de la ventana principal, en cualquier modo de operación que contiene la información de Log.	MFS03_FS01_F01	Device IDeviceWorker	PanelSimulation WindowLogView
		F02	Limpiar la ventana de Log de simulaciones anteriores por medio de un botón ubicado en cada una de las ventanas correspondientes.	MFS03_FS01_F02	Device IDeviceWorker	PanelSimulation WindowLogView
		F03	Exportar el Log de la Simulación en formato CSV por medio del botón ubicado en la ventana correspondiente en ambos modos de comunicación.	MFS03_FS01_F03	Device IDeviceWorker	PanelSimulation WindowLogView
		F04	Finalizar la aplicación manteniendo la configuración de la última simulación.	MFS03_FS01_F04		WindowSimbusMain MainClass
		F05	Devolver la configuración de Simulación a los valores por defecto en cualquier modo de operación a través de un botón específico ubicado en la ventana principal de la aplicación.	MFS03_FS01_F05		WindowSimbusMain PanelPonfigSimulation PanelDevMode PanelSimulation PanelQueryDetail
		F06	Representar en una tabla la configuración de los dispositivos Slave activos, mostrando Slave ID y estado en la ventana de Simulación.	MFS03_FS01_F06	Device IDeviceWorker	PanelSimulation PanelDevMode
		F07	Mostrar un panel de información general con el modo actual de	MFS03_FS01_F07		MainClass WindowSimbusMain

			comunicación, el tipo de comunicación, tipo y versión de Sistema Operativo y el botón de inicio y parada en la ventana de Simulación.			
		F08	Agregar dispositivos a la tabla de dispositivos configurados a través de un panel acoplado de configuración.	La característica 3 de todos los MFS anteriores, implementa implícitamente esta característica ya que la obtención de la información se hace desde dicho panel.		PanelDevMode WindowSimbusMain PanelSimulation

Fuente: *Autores del proyecto.*

La implementación de Simbus es realizada en el ambiente integrado de desarrollo MonoDevelop. El proyecto está estructurado en el árbol que se muestra en la figura 40. En este se pueden observar los cuatro proyectos que componen la solución Simbus. El primero muestra los paquetes que MonoDevelop genera para las diferentes plataformas; estos se describen a continuación:

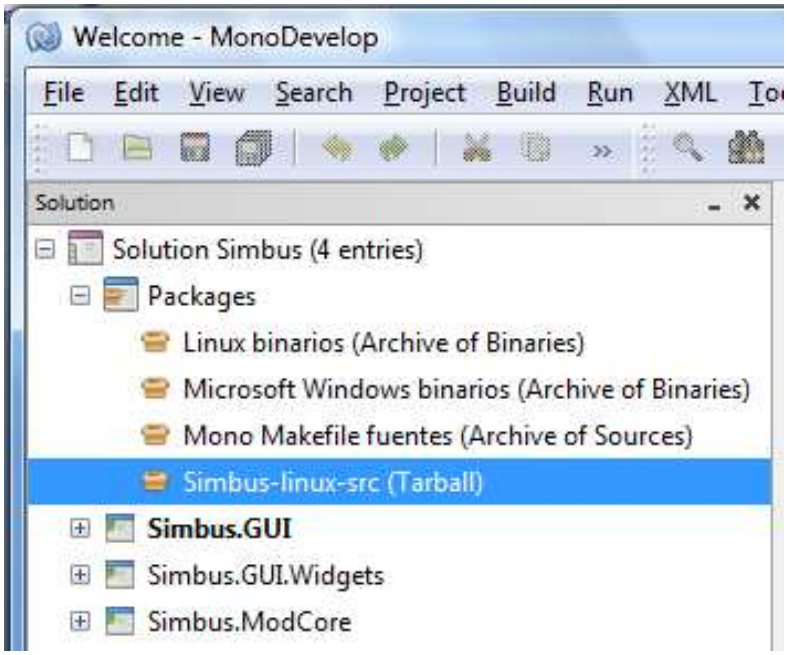
1. Linux Binarios (Archive of Binaries): archivo que contiene los binarios compilados para la ejecución de Simbus en entorno Linux.

2. Microsoft Windows binarios (Archive of Binaries): paquete zip que contiene los binarios compilados para la ejecución de Simbus en entorno Windows.

3. Mono Makefile fuentes (Archive of Sources): archivo con el árbol de proyecto.

4. Simbus – Linux - src (tarball): archivo con las fuentes previamente organizadas agregando scripts para compilar Simbus a través de la utilidad make[16].

Figura 40. Árbol de la solución Simbus.



Fuente: *Autores del proyecto.*

[16]make es una herramienta de generación o automatización de código para Unix/Linux.

El segundo proyecto que compone la solución Simbus es Simbus.GUI, dentro de este se encuentra la implementación de las clases y métodos incluidos en el diseño de la interfaz gráfica de usuario. El tercer proyecto Simbus.GUI.Widgets incluye la implementación de los paneles TCP Device y Serial Device, para utilizarlos como componentes personalizados en diferentes clases de Simbus.GUI. Por último el cuarto proyecto es Simbus.ModCore en este se implementan todas las clases y métodos funcionales, siendo este el núcleo principal de la solución.

Simbus.ModCore es un componente completamente independiente de los anteriores mencionados, desarrollado para que sobre él puedan ser implementadas diferentes interfaces gráficas. Para ésto ModCore ha sido completamente documentado en formato XML como se observa en la figura 41 y accesible mediante la opción Ayuda – Referencia biblioteca ModCore del menú principal de Simbus.

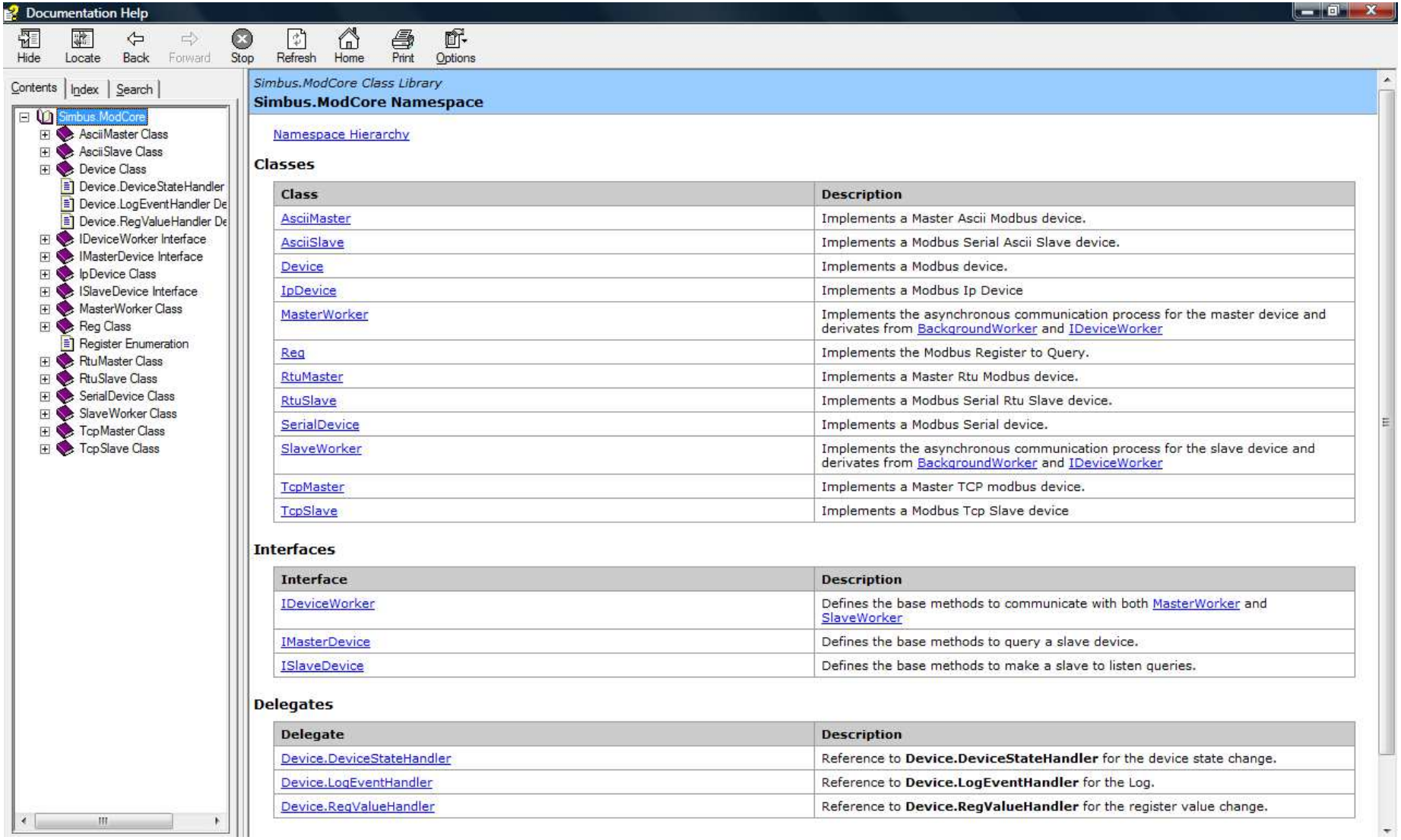
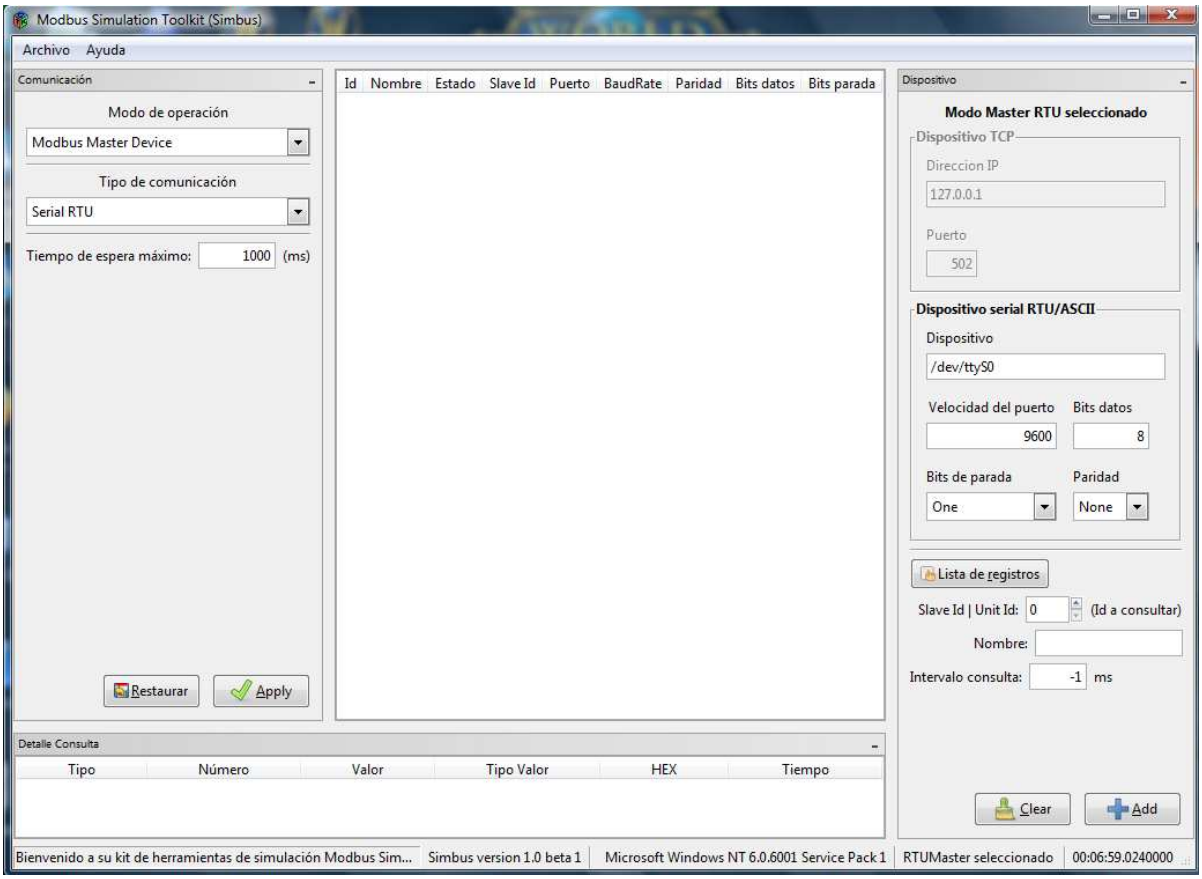


Figura 41. Documentación XML de Simbus.ModCore
Fuente: Autores *del proyecto.*

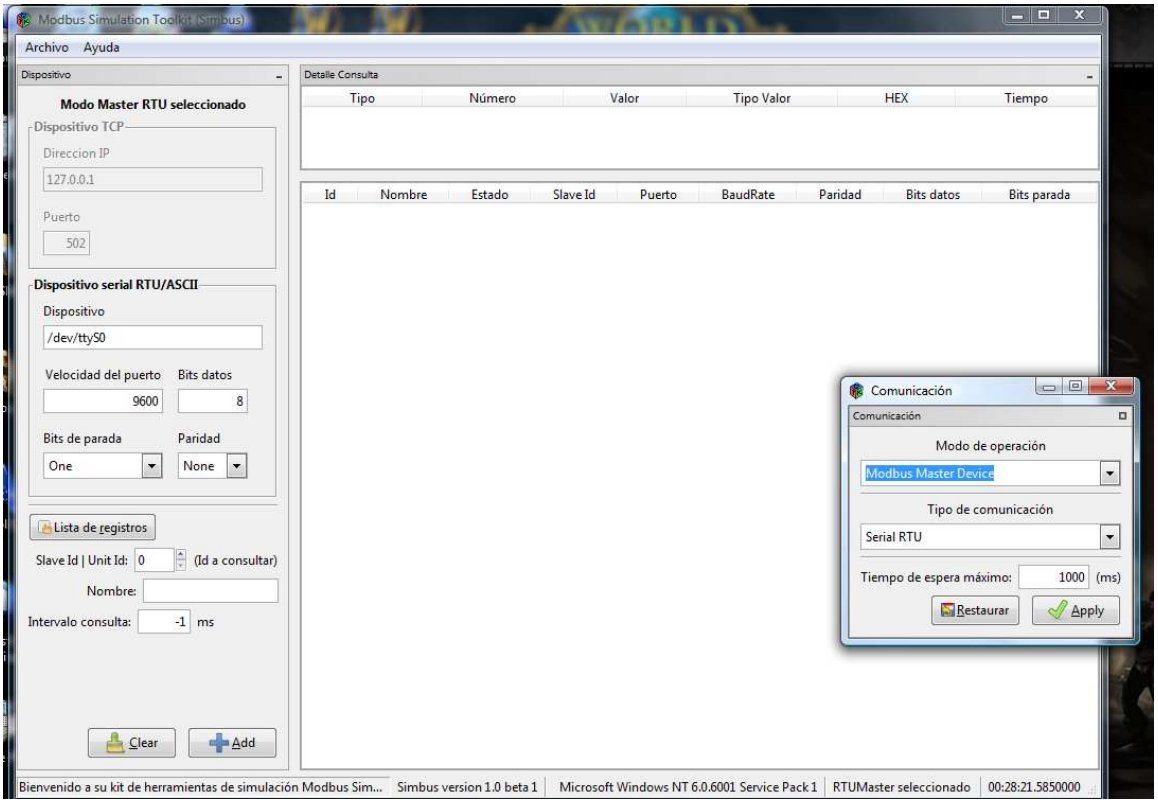
En cuanto a la interfaz gráfica de Simbus, en el diseño se tuvieron en cuenta la simplicidad y facilidad de uso. La interfaz cuenta con paneles acoplables, que permiten ser reubicados de forma personalizada o desacoplados de la ventana principal. En la figura 42, se puede observar la vista por defecto de Simbus y en la figura 43 se muestra una vista con los paneles reubicados y desacoplados.

Figura 42. Vista por defecto de Simbus



Fuente: *Autores del proyecto.*

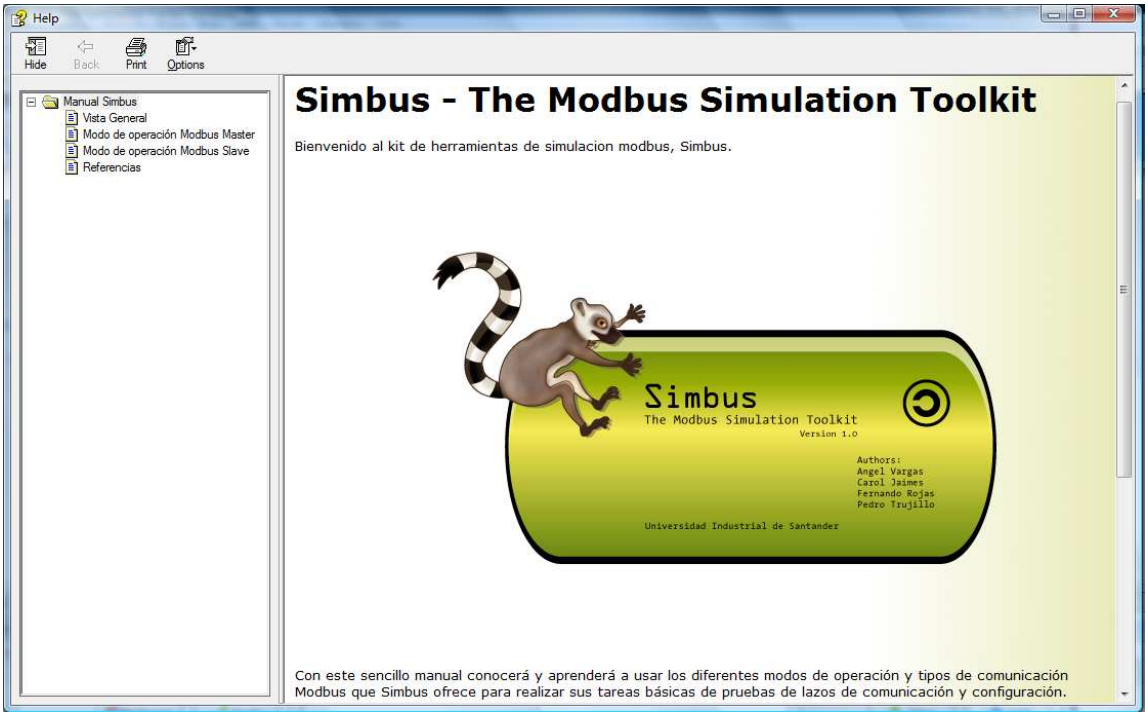
Figura 43. Vista paneles reubicados y desacoplados



Fuente: *Autores del proyecto.*

La aplicación también cuenta en el menú principal con acceso al manual de usuario y a la documentación de la librería Simbus.ModCore. En la figura 44, se muestra la vista del manual de usuario.

Figura 44. Manual de usuario



Fuente: *Autores del proyecto.*

6.2 PRUEBAS

6.2.1 PLAN DE PRUEBAS

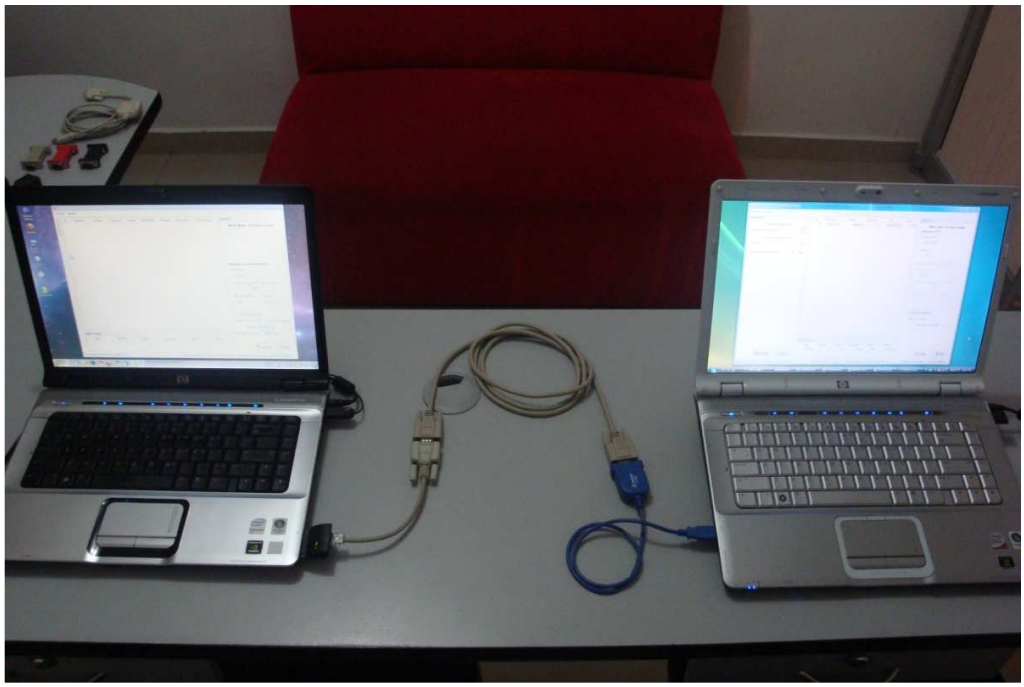
Para este proyecto se diseña un plan de pruebas específico que compruebe el correcto funcionamiento de cada una de las características implementadas. El diseño de este plan de pruebas se encuentra en el ANEXO 2 de este documento.

Las pruebas fueron realizadas en las instalaciones de SEAL LTDA, en un entorno controlado donde previamente se aseguró disponibilidad e integridad del lazo de comunicación tanto serial como TCP, de esta forma fue posible descartar problemas de funcionamiento alusivos a la comunicación física y concentrar esfuerzos en la verificación del correcto funcionamiento de Simbus.

En el caso de las pruebas de simulación Serial, tanto RTU como ASCII, se llevaron a cabo de simulador a simulador, es decir entre dos computadoras, cada una con Simbus instalado y siendo gestionado por un sistema operativo diferente. Las computadoras se conectaron directamente a través de un lazo de comunicación serial rs232[17], ver figura 45.

Las pruebas de comunicación TCP fueron realizadas estableciendo una comunicación con el sistema embebido multipropósito SGGT Versión 1.0[18] EP9302A ARM BASED[19], ver figura 46; ambos componentes se encontraban enlazados en la misma red de área local facilitando la conectividad entre los mismos, figura 47.

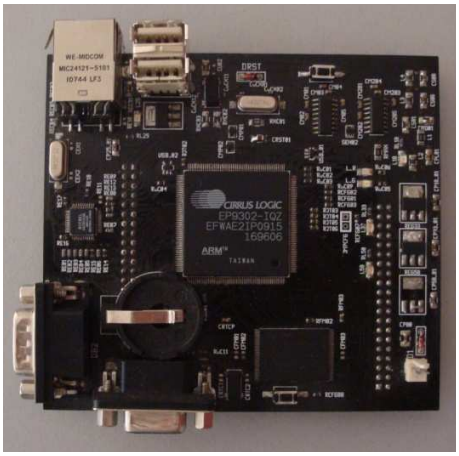
Figura 45. Montaje pruebas simulación serial RTU y ASCII.



Fuente: *Autores del proyecto.*

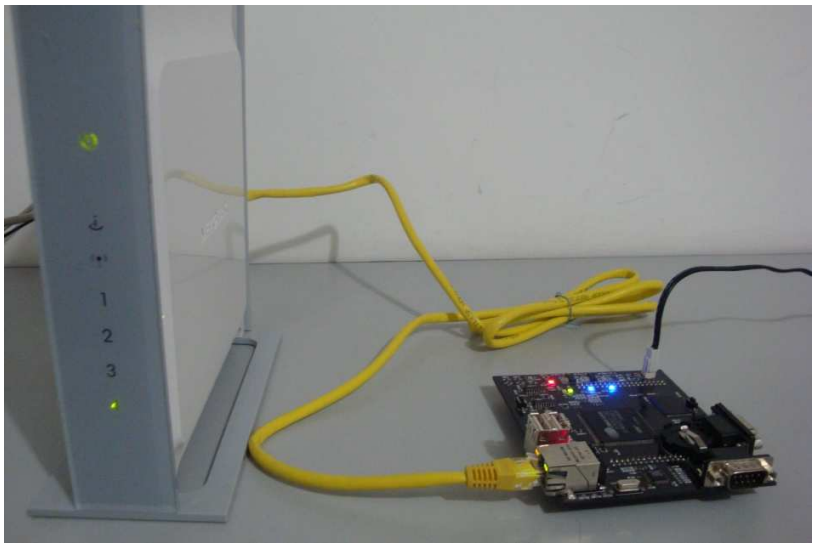
[17] Estándar rs232 para comunicación serial.
[18] Desarrollado SEAL LTDA. *http://www.sealibre.com/index.php*
[19] Advanced RISC Machine. Arquitectura de conjunto de instrucciones RISC desarrollada por ARM Holdings.

Figura 46. SGGT Versión 1.0 EP9302A ARM BASED



Fuente: *Autores del proyecto.*

Figura 47. SGGT Versión 1.0 conectada a la red de área local



Fuente: *Autores del proyecto.*

A continuación se presenta los resultados de la última aplicación del plan de pruebas a Simbus, las imágenes relacionadas con los resultados de la ejecución se encuentran en el anexo digital de este proyecto.

Tabla 9. Módulo de pruebas Modbus Master

Submódulo de operación	Prueba	Descripción	Criterio de Salida	Estado	Imagen Resultado
1.1 Simulación Master TCP	1.1.1 Correcta configuración del dispositivo.	Ingreso de los parámetros necesarios para configurar un dispositivo tipo Master TCP.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	A	P1_1_1
	1.1.2 Consulta en un registro tipo Coil.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cerró la conexión.	A	P1_1_2
	1.1.3Consulta en un registro tipo Contact.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cerró la conexión.	A	P1_1_3
	1.1.4 Consulta en un registro tipo Input.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cerró la conexión.	A	P1_1_4
	1.1.5 Consulta en un registro tipo Holding.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cerró la conexión.	A	P1_1_5
	1.1.6 Escritura en un registro tipo Coil con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza la escritura y cerró la conexión.	A	P1_1_6
	1.1.7 Escritura en un registro tipo Holding con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding y con un valor dados por el usuario.	Se conecta al dispositivo, realiza la escritura y cerró la conexión.	A	P1_1_7
	1.1.8 Escritura en un registro tipo Coil con un valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cerró la conexión.	A	P1_1_8
	1.1.9 Escritura en un registro tipo Holding con valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cerró la conexión.	A	P1_1_9
	1.1.10 Consulta en un registro tipo Coil, con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cerró la conexión a petición de usuario.	A	P1_1_10

	1.1.11Consulta en un registro tipo Contact con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_1_11
	1.1.12 Consulta en un registro tipo Input con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_1_12
	1.1.13 Consulta en un registro tipo Holding con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_1_13
	1.1.14 Escritura en un registro tipo Coil con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_1_14
	1.1.15 Escritura en un registro tipo Holding con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_1_15
	1.1.17 Escritura en un registro tipo Coil con un valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_1_16
	1.1.18 Escritura en un registro tipo Holding con valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_1_17
1.2 Simulación Master RTU	1.2.1 Correcta configuración del dispositivo.	Ingreso de los parámetros necesarios para configurar un dispositivo tipo Master TCP.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	A	P1_2_1
	1.2.2 Consulta en un registro tipo Coil.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo	Se conecta al dispositivo, realiza la consulta y cierra la	A	P1_2_2

		Coil dado por el usuario.	conexión.		
	1.2.3 Consulta en un registro tipo Contact.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	A	P1_2_3
	1.2.4 Consulta en un registro tipo Input.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	A	P1_2_4
	1.2.5 Consulta en un registro tipo Holding.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	A	P1_2_5
	1.2.6 Escritura en un registro tipo Coil con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	A	P1_2_6
	1.2.7 Escritura en un registro tipo Holding con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding y con un valor dados por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	A	P1_2_7
	1.2.8 Escritura en un registro tipo Coil con un valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	A	P1_2_8
	1.2.9 Escritura en un registro tipo Holding con valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	A	P1_2_9
	1.2.10 Consulta en un registro tipo Coil, con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_2_10
	1.2.11Consulta en un registro tipo Contact con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_2_11
	1.2.12 Consulta en un registro tipo Input con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_2_12
	1.2.13 Consulta en un registro tipo Holding con el intervalo de	Realiza múltiples consultas separadas por un intervalo de tiempo definido por	Se conecta al dispositivo, realiza múltiples consultas y	A	P1_2_13

	tiempo definido por el usuario.	el usuario a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	cierra la conexión a petición del usuario.		
	1.2.14 Escritura en un registro tipo Coil con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_2_14
	1.2.15 Escritura en un registro tipo Holding con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_2_15
	1.2.16 Escritura en un registro tipo Coil con un valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_2_16
	1.2.17 Escritura en un registro tipo Holding con valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_2_17
1.3 Simulación Master ASCII	1.3.1 Correcta configuración del dispositivo.	Ingreso de los parámetros necesarios para configurar un dispositivo tipo Master TCP.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	A	P1_3_1
	1.3.2 Consulta en un registro tipo Coil.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	A	P1_3_2
	1.3.3 Consulta en un registro tipo Contact.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	A	P1_3_3
	1.3.4 Consulta en un registro tipo Input.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	A	P1_3_4
	1.3.5 Consulta en un registro tipo Holding.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	A	P1_3_5
	1.3.6 Escritura en un registro tipo Coil con un valor dado por	Realiza una escritura a un dispositivo Slave en un número de registro de tipo	Se conecta al dispositivo, realiza la escritura y cierra la	A	P1_3_6

	el usuario.	Coil y con un valor dado por el usuario.	conexión.		
	1.3.7 Escritura en un registro tipo Holding con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding y con un valor dados por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	A	P1_3_7
	1.3.8 Escritura en un registro tipo Coil con un valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	A	P1_3_8
	1.3.9 Escritura en un registro tipo Holding con valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	A	P1_3_9
	1.3.10 Consulta en un registro tipo Coil, con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_3_10
	1.3.11 Consulta en un registro tipo Contact con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_3_11
	1.3.12 Consulta en un registro tipo Input con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_3_12
	1.3.13 Consulta en un registro tipo Holding con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	A	P1_3_13
	1.3.14 Escritura en un registro tipo Coil con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_3_14
	1.3.15 Escritura en un registro tipo Holding con un valor dado por el usuario con el intervalo de tiempo definido por el	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_3_15

	usuario.	con un valor dado por el usuario.			
	1.3.16 Escritura en un registro tipo Coil con un valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_3_16
	1.3.17 Escritura en un registro tipo Holding con valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	A	P1_3_17

Fuente: *Autores del proyecto.*

Tabla 10. Módulo de pruebas Modbus Slave

Submódulo de operación	Prueba	Descripcion	Criterio de Salida	Estado	Imagen Resultado
2.1 Simulación Slave TCP	2.1.1 Correcta configuración del dispositivo.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuariop.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	A	P2_1_1
	2.1.2 Escuchar y responder en un registro Modbus tipo Coil, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_1_2
	2.1.3 Escuchar y responder en un registro Modbus tipo Contact, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_1_3
	2.1.4 Escuchar y responder en un registro Modbus tipo Input, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_1_4

	2.1.5 Escuchar y responder en un registro Modbus tipo Holding, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_1_5
	2.1.6 Escuchar y responder en un registro Modbus tipo Coil, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_1_6
	2.1.7 Escuchar y responder en un registro Modbus tipo Contact, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_1_7
	2.1.8 Escuchar y responder en un registro Modbus tipo Input, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_1_8
	2.1.9 Escuchar y responder en un registro Modbus tipo Holding, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_1_9
2.2 Simulación Slave RTU	2.2.1 Correcta configuración del dispositivo.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuariop.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	A	P2_2_1
	2.2.2 Escuchar y responder en un registro Modbus tipo Coil, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_2_2
	2.2.3 Escuchar y responder en un registro Modbus tipo Contact, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_2_3
	2.2.4 Escuchar y responder en un registro Modbus tipo Input,	El dispositivo inicia correctamente el proceso de escucha con los	El dispositivo Modbus Master puede establecer la	A	P2_2_4

	con un valor de registro determinado por el usuario.	parámetros configurados; responde a la solicitud Modbus con el registro configurado.	comunicación y obtener el valor especificado por el usuario.		
	2.2.5 Escuchar y responder en un registro Modbus tipo Holding, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_2_5
	2.2.6 Escuchar y responder en un registro Modbus tipo Coil, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_2_6
	2.2.7 Escuchar y responder en un registro Modbus tipo Contact, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_2_7
	2.2.8 Escuchar y responder en un registro Modbus tipo Input, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_2_8
	2.2.9 Escuchar y responder en un registro Modbus tipo Holding, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_2_9
2.3 Simulación Slave ASCII	2.3.1 Correcta configuración del dispositivo.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuariop.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	A	P2_3_1
	2.3.2 Escuchar y responder en un registro Modbus tipo Coil, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_3_2
	2.3.3 Escuchar y responder en un registro Modbus tipo Contact, con un valor de registro determinado por el	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_3_3

	usuario.	configurado.			
	2.3.4 Escuchar y responder en un registro Modbus tipo Input, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_3_4
	2.3.5 Escuchar y responder en un registro Modbus tipo Holding, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener el valor especificado por el usuario.	A	P2_3_5
	2.3.6 Escuchar y responder en un registro Modbus tipo Coil, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_3_6
	2.3.7 Escuchar y responder en un registro Modbus tipo Contact, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_3_7
	2.3.8 Escuchar y responder en un registro Modbus tipo Input, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_3_8
	2.3.9 Escuchar y responder en un registro Modbus tipo Holding, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master puede establecer la comunicación y obtener un valor generado aleatoriamente.	A	P2_3_9

Fuente: *Autores del proyecto.*

Tabla 11. Módulo Integridad

Submódulo de operación	Prueba	Descripcion	Criterio de Salida	Estado	Imagen Resultado
3.1 Comunicación TCP	3.1.1 Prueba de lazo entre aplicaciones Simbus.	Simbus en modo Master en un computador 1 consulta a Simbus Slave en un computador 2. Para un registro de cada tipo simultáneamente.	Se establece la comunicación entre ambas partes y la información enviada por Simbus Slave es visualizada correctamente por Simbus Master.	A	P3_1_1
3.2 Comunicación RTU	3.2.1 Prueba de lazo entre aplicaciones Simbus.	Simbus en modo Master en un computador 1 consulta a Simbus Slave en un computador 2. Para un registro de cada tipo simultáneamente.	Se establece la comunicación entre ambas partes y la información enviada por Simbus Slave es visualizada correctamente por Simbus Master.	A	P3_2_1
3.3. Comunicación ASCII	3.3.1 Prueba de lazo entre aplicaciones Simbus.	Simbus en modo Master en un computador 1 consulta a Simbus Slave en un computador 2. Para un registro de cada tipo simultáneamente.	Se establece la comunicación entre ambas partes y la información enviada por Simbus Slave es visualizada correctamente por Simbus Master.	A	P3_3_1

Fuente: *Autores del proyecto.*

7. RECOMENDACIONES

- Desarrollar aplicaciones que atiendan otros protocolos de comunicación industrial importantes tales como Profibus[20] y Fieldbus[21].

- Lograr que la aplicación sea capaz de simular dispositivos en sus diferentes modos simultáneamente.

- Expandir las características de la simulación del dispositivo para incluir funcionalidades diferentes a la comunicación tales como la generación de señales provenientes de sensores e instrumentos.

- Agregar al sistema de consultas del dispositivo Maestro la posibilidad de identificar consultas contiguas y combinarlas en una única consulta Modbus.

- Incluir soporte a través del uso de las librerías gettext[22] para varios idiomas, es decir internacionalizar la interfaz grafica de usuario.

[20] Profibus es un estándar de comunicaciones para bus de campo. Deriva de las palabras PROcess FIeld BUS.
[21] Fieldbus es el nombre de una familia de protocolos parea redes de computadores industriales usado en sistemas de control distribuido en tiempo real, ahora estandarizado como IEC 61158.
[22] gettext es la biblioteca GNU de internacionalización (i18n). Comúnmente es usada para escribir programas con interfaz en múltiples idiomas.

8. CONCLUSIONES

- A través del uso de la metodología ágil FDD, se logró desarrollar la aplicación cumpliendo con las diferentes etapas descritas por la misma, adquiriendo la habilidad de desarrollar software de manera metódica y con procesos que garanticen calidad y eficiencia. Aunque está diseñada para trabajar con grupos de mediana o gran cantidad y orientada a software para negocio, dada su versatilidad se ha podido adaptar a un grupo pequeño y al campo técnico.

- Con el adecuado acompañamiento del Lenguaje Unificado de Modelado UML la metodología, se logra describir completa y correctamente cada proceso necesario para la construcción de este proyecto.

- Aunque el conjunto de librerías gráficas GTK no sea el más amigable y carezca de una gran paleta de componentes gráficos, su capacidad de soportar múltiples plataformas y desarrollar aplicaciones de rápida respuesta al usuario inclinan ampliamente la balanza a su favor.

- Después de este ejercicio de trabajo colaborativo se reafirmó completamente la necesidad imperante de trabajar con un sistema de control de versiones para llevar un correcto registro y control de los cambios realizados, no solo para el código fuente, sino además para la documentación.

- Con el desarrollo de Simbus se logró dar una solución a un problema real, en un proceso que entrelazó lo teórico y lo práctico.

- Este proyecto por su característica de código abierto y de libre distribución fue posible gracias a la colaboración de toda una comunidad internacional.

- La selección y utilización del lenguaje de programación C#, orientado a objetos con una rápida curva de aprendizaje y a su vez robusto, permitió que el proyecto se llevara a cabo cumpliendo a cabalidad con los objetivos y en los tiempos estimados.

- Por último gracias a la realización de este proyecto se logró ampliar los conocimientos en diversas áreas de acción para la Ingeniería de Sistemas.

BIBLIOGRAFÍA

AKADIA - Global Competence in Today's Information Technology. *Delegates and events in c# / .net* [en línea]. Disponible en: <http://www.akadia.com/services/dotnet_delegates_and_events.html>

BOYER Stuart A. SCADA Supervisory Control and Data Acquisition. United States of America: Instrument Society of America, 1999. 201 p.

GNU Project. *Licenses* [en línea]. GNU Project – Free Software Foundation, 2010 [fecha consulta: Durante todo el desarrollo] Disponible en: < http://www.gnu.org/ >

GTK Project. *Documentation* [en línea]. GTK Project, 2010 [fecha de consulta: Durante todo el desarrollo] Disponible en: < http://www.gtk.org/ >

MSDN Microsoft Developer Network. *MSDN Library* [en línea]. MSDN Microsoft Developer Network, 2010 [fecha de consulta: Durante todo el desarrollo] <http://msdn.microsoft.com/en-us/library/default.aspx>

MONO Project. *Documentation* [en línea: Durante todo el desarrollo]. Mono Project, 2010 [fecha de consulta: Durante todo el desarrollo] Disponible en: < http://www.mono-project.com/Main_Page >

MONODEVELOP. *Documentation* [en línea: Durante todo el desarrollo]. MonoDevelop, 2010 [fecha de consulta: Durante todo el desarrollo]Disponible en: < http://monodevelop.com >

MODICON. Modbus Reference Guide. North Andover, Massachusetts: MODICON Inc. Industrial Automation Systems, 1996, 115 p.

MODBUS IDA. Modbus Application Protocol Specification V1.1b. United states of America: Modbus-Ida, 2006, 51 p.

THE CODE PROJECT. *MVC, Model View Controller Using C#, Delegates and Events in .NET* [en línea]. The Code Project, 2010 [fecha de consulta : Diciembre de 2009] Disponible en: <http://www.codeproject.com/KB/cs/model_view_controller.aspx >

PALMER R. Stephen, FELSING M. John. A Practical Guide to Feature-Driven Development. Prentice Hall. Upper Saddle River NJ, United States 2002.

SCOTT Alexander, NModbus is a C# 3.0 implementation of the Modbus protocol [en línea]. NMobus, 2009. Disponible en: <http://nmodbus.com/Default.aspx>.

SIMPLY MODBUS. DATA COMMUNICATION TEST SOFTWARE. *Documentation* [en línea]. Simply Modbus, 2010. [fecha de consulta: Febrero 15 de 2009]Disponible en: <http://www.simplymodbus.ca/index.html>.

SUSSMAN, Ben Collins, FITZPATRICK, Brian W. y PILATO, C. Michael. Version *Control with Subversion, For Subversion 1.1. PDF* [en línea]. Disponible en: < http://svnbook.red-bean.com/en/1.1/svn-book.pdf >

ANEXOS

ANEXO 1: GLOSARIO DE TÉRMINOS

Protocolo: En informática, un protocolo es un conjunto de reglas usadas por computadoras para comunicarse unas con otras a través de una red. Un protocolo es una convención o estándar que controla o permite la conexión, comunicación, y transferencia de datos entre dos puntos finales. En su forma más simple, un protocolo puede ser definido como las reglas que dominan la sintaxis, semántica y sincronización de la comunicación. Los protocolos pueden ser implementados por hardware, software, o una combinación de ambos. A su más bajo nivel, un protocolo define el comportamiento de una conexión de hardware.

Modbus: Modbus es un protocolo de comunicación desarrollado por sistemas Modicon, en términos simples, es una forma de enviar información entre dispositivos electrónicos.

Modicon: En 1968 con un grupo inicial de ingenieros, Richard E. Morley fundó Bedford Associates e inventó el primer controlador lógico programable o PLC. Él estableció la compañía Modicon, nombre que deriva de Modular Digital Control. La marca Modicon ha tenido numerosos propietarios y hoy en día es propiedad de Schneider Electric.

Maestro (Master): Dispositivo único dentro de una red Modbus que puede iniciar transacciones llamadas consultas (queries).

Esclavo (Slaves): Dispositivos dentro de una red Modbus que responden las peticiones del Maestro, enviando al maestro los datos requeridos por este, o realizando la acción dada en la consulta.

Red Modbus: Red de comunicaciones entre dispositivos basada en el protocoo Mobdus.

Protocolo Abierto: aquel en el cual sus características son de libre acceso, tanto a empresas como a usuarios, los cuales a su vez pueden obtener la suficiente documentación para su implementación.

Automatización industrial: (automatización; del griego antiguo auto: guiado por uno mismo) es el uso de sistemas o elementos computarizados para controlar maquinarias y/o procesos industriales substituyendo a operadores humanos.

Controladores Lógicos Programables PLCs: Los PLC (Programmable Logic Controller en sus siglas en inglés)o Controlador de lógica programable, son dispositivos electrónicos muy usados en Automatización Industrial. Un PLC es un

hardware industrial, que se utiliza para la obtención de datos. Una vez obtenidos, los pasa a través de bus (por ejemplo por Ethernet) en un servidor.

Instrumentación industrial: Rama de la ciencia que trata de la medición y control del conjunto de instrumentos de una planta o proceso. Dentro de la ingeniería multidisciplinaria, es la disciplina que se encarga de toda la instrumentación. En estos casos incluye instrumentación como control de procesos, pero también otros temas como comunicaciones, telefonía, sistemas de vídeo, redes de computadores y otros.

Coils: Nombre dado a los valores discretos en las tablas de almacenamiento del estándar Modbus.

Registers: Nombre dado a los valores numéricos en las tablas de almacenamiento del estándar Modbus.

ID del Esclavo: Número de dirección único asignado a cada uno de los dispositivos Esclavos (Slaves) en una red. Su valor está entre 1 y 247.

Offset: Diferencia entre cada los número únicos que tiene cada uno de los Coils o Registers en las tablas de almacenamiento del estándar Modbus.

Función Modbus: Acción que el maestro pide al esclavo que realice, ya sea de lectura o escritura.

Código de función Modbus: Este número dice al esclavo cual tabla de almacenamiento acceder y si leer de ella o escribir sobre la misma.

Modos de transmisión Modbus: Formas o modos de transmitir a información dentro de una red Modbus.

Serial ASCII: Modo de transmisión serial, donde cada byte de 8 bits en un mensaje es enviado como dos caracteres ASCII.

Serial RTU: Modo de transmisión seria, donde cada byte de 8 bits en un mesaje contiene dos caracteres hexadecimales de 4 bits.

Modbus TCP: Modo de transmisión que envía el mensaje Modbus a través de protocolo TCP/IP,

PDU: Unidad de datos del protocolo, resultado de remover a un mensaje Modbus RTU la ID del esclavo y el campo CRC.

ADU: Application Data Unit o unidad de datos Modbus general, donde el trazado del protocolo sobre buses específicos o redes pude introducir algunos campos adicionales a la PDU dando como resultado la ADU.

MBAP: Es un encabezado dedicado es usado sobre TCP/IP para identificar la unidad de datos ADU.

SCADA: SCADA (Supervisory Control And Data Adquisition) o en español registro de datos y control de supervisión es principalmente un sistema que monitoriza y controla un proceso ya sea industrial, de infraestructura y de Instalaciones.

HMI: Interfaz hombre-máquina o HMI es el aparato o software que presenta los datos procesados al operador (humano), y que le permite al mismo controlar el proceso. El termino HMI es usado principalmente para referirse a interfaz de usuarios (User Interface) de sistemas mecánicos o electromecánicos.

Nmodbus: NModbus es una biblioteca de libre distribución que implementa el protocolo modbus en el lenguaje C# 3.0. Soporta los protocolos serial ASCII, serial RTU, serial sobre USB ASCII, serial sobre USB RTU, TCP y UDP.

Licencia Pública General GNU: La Licencia Pública General GNU, por sus siglas en inglés General Public License, o simplemente su acrónimo en inglés GNU GPL es un tipo de licencia creada en la mitad de la década de los 80 por la Free Software Foundation. Su fin es el de proteger la distribución, modificación y uso de software.

Software Libre: El software libre (en inglés free software, aunque en realidad esta denominación también puede significar gratis, y no necesariamente libre, por lo que se utiliza el hispanismo libre software también en inglés) es la denominación del software que respeta la libertad de los usuarios sobre su producto adquirido y, por tanto, una vez obtenido puede ser usado, copiado, estudiado, cambiado y redistribuido libremente.

Free Software Foundation: La Free Software Foundation (Fundación para el software libre) es una organización creada en octubre de 1985 por Richard Stallman y otros entusiastas del software libre con el propósito de difundir este movimiento. La Fundación para el software libre (FSF) se dedica a eliminar las restricciones sobre la copia, redistribución, entendimiento, y modificación de programas de computadoras. Con este objeto, promociona el desarrollo y uso del software libre en todas las áreas de la computación, pero muy particularmente, ayudando a desarrollar el sistema operativo GNU.

Mono: Mono es una plataforma de software diseñada para permitir a los desarroladores crear aplicaciones para diferentes plataformas. Patrocinada por Novell, Mono es una implementación código abierto del Framework Microsoft .NET basado en los estándares ECMA para C# y el Common Language Runtime.

Mono Develop: Mono es un ambiente integrado de desarrollo (IDE, Integrated Development Enviroment) diseñado para C# y otros lenguajes .NET.

Subversion: Subversion SVN es un sistema de control de versiones de código abierto. Esto es, Subversion maneja archivos y directorios en el tiempo.

ANEXO 2: DISEÑO PLAN DE PRUEBAS

I. INTRODUCCIÓN

El plan de pruebas busca cubrir las pruebas de funcionamiento unitario por conjuntos de características y a su vez el funcionamiento global de la aplicación. Las pruebas si dividen en dos módulos llamados Pruebas Modo Operación Master y Pruebas Modo Operación Slave, dentro de estos dos módulos se realizan pruebas para verificar el correcto y completo funcionamiento de cada conjunto de características implementado.

II. PRUEBAS POR MÓDULOS

Cada conjunto de pruebas detallado en la tabla muestra una descripción de la funcionalidad que debe tener cada uno de los módulos y la salida esperada en cada prueba.

El estado "aceptado" (A) de la prueba implica que se cumplió con el criterio de salida definido, el estado "rechazado" (R) implica que no se cumplió con el criterio de salida definido por fallos mayores, y el estado "pendiente" (P) implica que alguna de las pruebas que componen el módulo no cumplió a cabalidad con el criterio de salida definido y que permanece en este estado mientras todos los criterios de salida que lo componen sean cumplidos, un módulo puede estar en "pendiente" mientras tenga el 50% o más de sus pruebas aceptadas.

1. SIMULACIÓN MODBUS MASTER

Submódulo de operación	Prueba	Descripción	Criterio de Salida	Estado
1.1 Simulación Master TCP	1.1.1 Correcta configuración del dispositivo.	Ingreso de los parámetros necesarios para configurar un dispositivo tipo Master TCP.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	
	1.1.2 Consulta en un registro tipo Coil.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.1.3Consulta en un registro tipo Contact.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.1.4 Consulta en un registro tipo Input.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.1.5 Consulta en un registro tipo Holding.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.1.6 Escritura en un registro tipo Coil con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.1.7 Escritura en un registro tipo Holding con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding y con un valor dados por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.1.8 Escritura en un registro tipo Coil con un valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.1.9 Escritura en un registro tipo Holding con valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.1.10 Consulta en un registro tipo Coil, con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza multiples consultas y cerró la conexión a petición de usuario.	
	1.1.11Consulta en un registro tipo Contact con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	

		de tipo Contact dado por el usuario.		
	1.1.12 Consulta en un registro tipo Input con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.1.13 Consulta en un registro tipo Holding con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.1.14 Escritura en un registro tipo Coil con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
	1.1.15 Escritura en un registro tipo Holding con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
	1.1.16 Escritura en un registro tipo Coil con un valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
	1.1.17 Escritura en un registro tipo Holding con valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
1.2 Simulación Master RTU	1.2.1 Correcta configuración del dispositivo.	Ingreso de los parámetros necesarios para configurar un dispositivo tipo Master TCP.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	
	1.2.2 Consulta en un registro tipo Coil.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cerró la conexión.	
	1.2.3 Consulta en un registro tipo Contact.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.2.4 Consulta en un registro tipo Input.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Input dado por	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	

		el usuario.		
	1.2.5 Consulta en un registro tipo Holding.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.2.6 Escritura en un registro tipo Coil con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.2.7 Escritura en un registro tipo Holding con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding y con un valor dados por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.2.8 Escritura en un registro tipo Coil con un valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.2.9 Escritura en un registro tipo Holding con valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.2.10 Consulta en un registro tipo Coil, con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.2.11Consulta en un registro tipo Contact con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.2.12 Consulta en un registro tipo Input con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.2.13 Consulta en un registro tipo Holding con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.2.14 Escritura en un registro tipo Coil con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
	1.2.15 Escritura en un registro tipo Holding con un valor dado por el usuario con el intervalo de tiempo	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	

	definido por el usuario.	de tipo Holding y con un valor dado por el usuario.		
	1.2.16 Escritura en un registro tipo Coil con un valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
	1.2.17 Escritura en un registro tipo Holding con valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
1.3 Simulación Master ASCII	1.3.1 Correcta configuración del dispositivo.	Ingreso de los parámetros necesarios para configurar un dispositivo tipo Master TCP.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	
	1.3.2Consulta en un registro tipo Coil.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.3.3 Consulta en un registro tipo Contact.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.3.4 Consulta en un registro tipo Input.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.3.5 Consulta en un registro tipo Holding.	Realiza una consulta a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza la consulta y cierra la conexión.	
	1.3.6 Escritura en un registro tipo Coil con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.3.7 Escritura en un registro tipo Holding con un valor dado por el usuario.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding y con un valor dados por el usuario.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.3.8 Escritura en un registro tipo Coil con un valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	
	1.3.9Escritura en un registro tipo Holding con valor generado aleatoriamente.	Realiza una escritura a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario y con un valor generado	Se conecta al dispositivo, realiza la escritura y cierra la conexión.	

		aleatoriamente.		
	1.3.10 Consulta en un registro tipo Coil, con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.3.11 Consulta en un registro tipo Contact con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Contact dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.3.12 Consulta en un registro tipo Input con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Input dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.3.13 Consulta en un registro tipo Holding con el intervalo de tiempo definido por el usuario.	Realiza múltiples consultas separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding dado por el usuario.	Se conecta al dispositivo, realiza múltiples consultas y cierra la conexión a petición del usuario.	
	1.3.14 Escritura en un registro tipo Coil con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
	1.3.15 Escritura en un registro tipo Holding con un valor dado por el usuario con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor dado por el usuario.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
	1.3.16 Escritura en un registro tipo Coil con un valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Coil y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	
	1.3.17 Escritura en un registro tipo Holding con valor generado aleatoriamente y con el intervalo de tiempo definido por el usuario.	Realiza múltiples escrituras separadas por un intervalo de tiempo definido por el usuario a un dispositivo Slave en un número de registro de tipo Holding y con un valor generado aleatoriamente.	Se conecta al dispositivo, realiza múltiples escrituras y cierra la conexión a petición del usuario.	

2. SIMULACIÓN MODBUS SLAVE

Submódulo de operación	Prueba	Descripcion	Criterio de Salida	Estado
2.1 Simulación Slave TCP	2.1.1 Correcta configuración del dispositivo.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuariop.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	
	2.1.2Escuchar y responder en un registro Modbus tipo Coil, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.1.3 Escuchar y responder en un registro Modbus tipo Contact, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.1.4 Escuchar y responder en un registro Modbus tipo Input, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.1.5 Escuchar y responder en un registro Modbus tipo Holding, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.1.6 Escuchar y responder en un registro Modbus tipo Coil, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.1.7 Escuchar y responder en un registro Modbus tipo Contact, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.1.8 Escuchar y responder en un registro Modbus tipo Input, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.1.9 Escuchar y responder en un registro Modbus tipo Holding, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
2.2. Simulación	2.2.1 Correcta configuración del dispositivo.	El dispositivo se agrega a la lista, la información se visualiza correctamente	El dispositivo se agrega a la lista, la información se visualiza	

Slave RTU		con los parámetros configurados por el usuariop.	correctamente con los parámetros configurados por el usuario.	
	2.2.2Escuchar y responder en un registro Modbus tipo Coil, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.2.3 Escuchar y responder en un registro Modbus tipo Contact, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.2.4 Escuchar y responder en un registro Modbus tipo Input, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.2.5 Escuchar y responder en un registro Modbus tipo Holding, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.2.6 Escuchar y responder en un registro Modbus tipo Coil, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.2.7 Escuchar y responder en un registro Modbus tipo Contact, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.2.8 Escuchar y responder en un registro Modbus tipo Input, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.2.9 Escuchar y responder en un registro Modbus tipo Holding, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
2.3 Simulación Slave ASCII	2.3.1 Correcta configuración del dispositivo.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuariop.	El dispositivo se agrega a la lista, la información se visualiza correctamente con los parámetros configurados por el usuario.	
	2.3.2 Escuchar y responder en un registro Modbus tipo Coil, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	

	2.3.3 Escuchar y responder en un registro Modbus tipo Contact, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.3.4 Escuchar y responder en un registro Modbus tipo Input, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.3.5 Escuchar y responder en un registro Modbus tipo Holding, con un valor de registro determinado por el usuario.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener el valor especificado por el usuario.	
	2.3.6 Escuchar y responder en un registro Modbus tipo Coil, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.3.7 Escuchar y responder en un registro Modbus tipo Contact, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.3.8 Escuchar y responder en un registro Modbus tipo Input, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	
	2.3.9 Escuchar y responder en un registro Modbus tipo Holding, con un valor generado aleatoriamente.	El dispositivo inicia correctamente el proceso de escucha con los parámetros configurados; responde a la solicitud Modbus con el registro configurado.	El dispositivo Modbus Master pudo establecer la comunicación y obtener un valor generado aleatoriamente.	

3. INTEGRIDAD

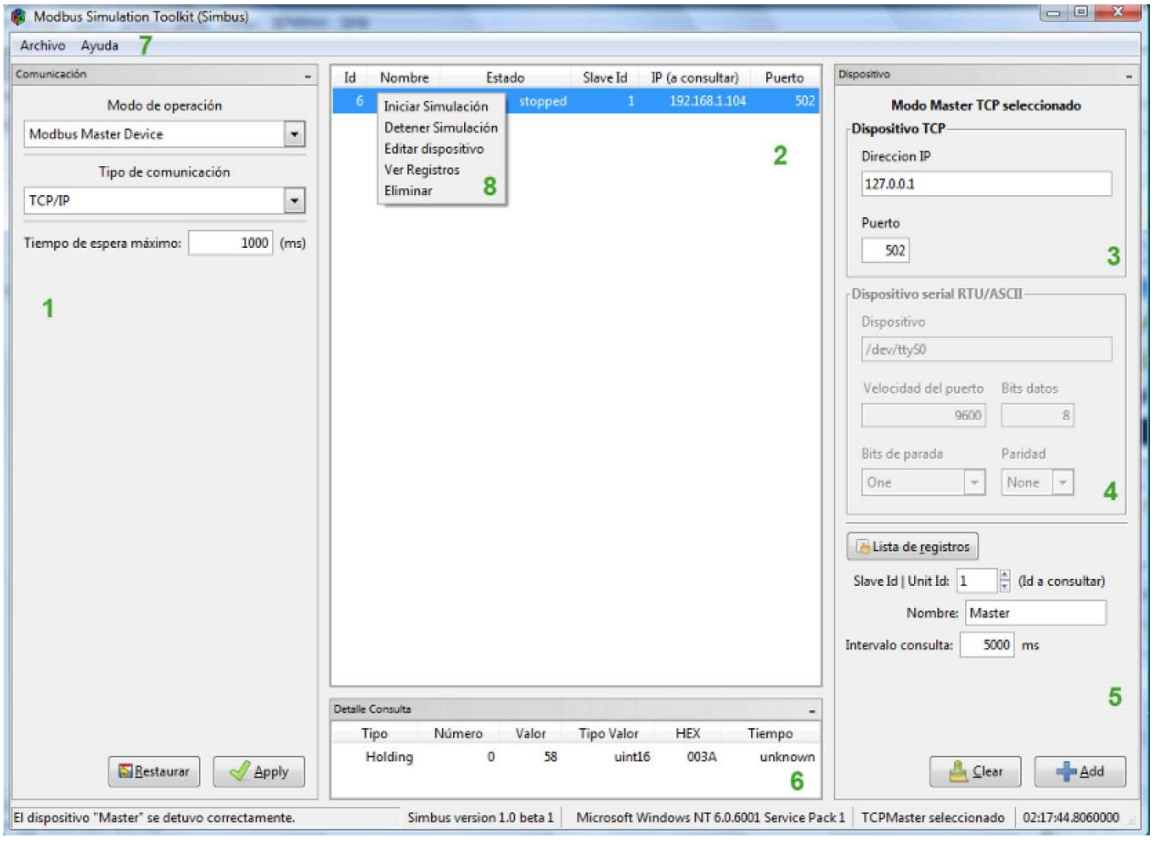
Submódulo de operación	Prueba	Descripcion	Criterio de Salida	Estado
3.1 Comunicación TCP	3.1.1 Prueba de lazo entre aplicaciones Simbus.	Simbus en modo Master en un computador 1 consulta a Simbus Slave en un computador 2. Para un registro de cada tipo simultáneamente.	Se establece la comunicación entre ambas partes y la información enviada por Simbus Slave es visualizada correctamente por Simbus Master.	
3.2 Comunicación RTU	3.2.1 Prueba de lazo entre aplicaciones Simbus.	Simbus en modo Master en un computador 1 consulta a Simbus Slave en un computador 2. Para un registro de cada tipo simultáneamente.	Se establece la comunicación entre ambas partes y la información enviada por Simbus Slave es visualizada correctamente por Simbus Master.	
3.3. Comunicación ASCII	3.3.1 Prueba de lazo entre aplicaciones Simbus.	Simbus en modo Master en un computador 1 consulta a Simbus Slave en un computador 2. Para un registro de cada tipo simultáneamente.	Se establece la comunicación entre ambas partes y la información enviada por Simbus Slave es visualizada correctamente por Simbus Master.	

ANEXO 3. RESUMEN MANUAL DE USUARIO

Simbus - The Modbus Simulation Toolkit
Bienvenido al kit de herramientas de simulacion modbus, Simbus.

Con este sencillo manual conocerá y aprenderá a usar los diferentes modos de operación y tipos de comunicación
Modbus que Simbus ofrece para realizar sus tareas básicas de pruebas de lazos de comunicación y configuración.
Así mismo se le ofrecerá una referencia para aprender más acerca del protocolo de comunicación base y a los principales proyectos que hacen posible esta solución.
Esperamos que disfrute usando Simbus tanto como nosotros disfrutamos desarrollandola.
Atentamente. El equipo de desarrollo.
Simbus es distribuida bajo los términos de la licencia GPL versión 2.

Vista general
La vista principal de Simbus contiene los siguientes elementos.



1. Panel Comunicación
El panel de comunicación permite seleccionar los diferentes modos de operación que se listan a continuación:
a. Modbus Master Device
b. Modbus Serial Device
Para cada modo de operación se encuentran disponibles los siguientes tipos de operación:
a. TCP/IP
b. Serial RTU
c. Serial ASCII
Finalmente es posible asignar el tiempo máximo de espera entre solicitudesya sea de lectura o de escritura en dispositivos Master o Slave. Para cambiar la vista general al modo de simulación seleccionado se debe presionar el botón Apply o Aplicar. Para volver a los valores por defecto presione el botón Restaurar.

2. Lista de dispositivos
La lista de dispositivos muestra los dispositivos actualmente configurados. Los estados de los dispositivos,
así como la configuración básica de su comunicación son visualizados en esta lista.
Para ejecutar tareas sobre los dispositivos configurados se accede al menú contextual descrito en el numeral 8.

3. Panel Dispositivo TCP
El panel de dispositivo TCP es activado al seleccionar cualquier modo de operación que incluya un tipo de comunicación TCP. Este panel permite configurar la dirección IP y el número de puerto que gestiona la comunicación.

4. Panel Dispositivo Serial RTU/ASCII
El panel de Dispositivo Serial es activado al seleccionar cualquier modo de operación que incluya un tipo de comunicación serial. Este panel permite seleccionar el puerto y configurar los parámetros de comunicación serial.

5. Panel Dispositivo
El panel Dispositivo permite lanzar el diálogo de configuración de los registros a través del botón lista de registros, además permite seleccionar la Slave Id del dispositivo, dar un nombre y asignar el tiempo entre consultas si se trata de un dispositivo Master. El botón Add permite agregar el dispositivo a la Lista de Dispositivos configurados y el botón Clear limpia la Lista de Registros y devuelve el panel a sus valores por defecto incluyendo el panel interno que se encuentra activo.

6. Panel Detalle Consulta
El panel Detalle Consulta muestra los registros del dispositivo seleccionado mediante doble clic en la Lista de Dispositivos. También responde al evento en que cambie el valor de un registro de esta manera mantiene la lista de registros con los valores actuales.

7. Menú principal Simbus
El menú principal de Simbus cuenta con las siguientes opciones:
a. Archivo - Nueva simulación: inicia una nueva simulación en modo por defecto.
b. Archivo - Salir: Finaliza la aplicación, requiere la confirmación del usuario.
c. Ayuda - Acerca: Muestra la versión que se está ejecutando, los autores de este proyecto y el tipo de licencia de Simbus.
d. Ayuda - Manual de usuario: Despliega este manual.
e. Ayuda - Restaurar vista: Cierra y devuelve todos los paneles de la vista principal de Simbus
a su configuración por defecto.

8. Menú contextual Lista de dispositivos

El menú contextual muestra las tareas a ejecutar sobre los dispositivos configurados. Iniciar Simulación,
Detener Simulación, Editar dispositivo, Ver Registros y Eliminar. Editar dispositivo permite cambiar la configuración del dispositivo seleccionado, Ver registros muestra los registros configurados para el dispositivo y por último eliminarlo de la lista mediante la opción Eliminar.

Modo de operación Modbus Master

Al configurar una nueva simulación en modo Modbus Master, Simbus le permite seleccionar el tipo de comunicación y el tiempo de espera máximo, en el panel Comunicación. Es necesario dar clic al botón Apply para ajustar la vista general con la configuración seleccionada.

Master TCP

En este caso Simbus habilita el panel de dispositivo TCP. Seleccione la dirección IP y el número de puerto del Slave a consultar. Haciendo clic en el botón Lista de Registros del panel de Dispositivo visualice el diálogo que permite
configurar los registros a consultar. Haciendo clic derecho sobre el diálogo de registros configurados, despliegue el menú contextual y seleccione la acción añadir registro.

Seleccione el Tipo de registro a consultar, el número de registro y si lo requiere active la opción de escribir, dando un valor o un rango para generar el valor a escribir aleatoriamente. Al dar clic en OK, inmediatamente el registro es agregado en la Lista de Registros. Esta muestra el tipo de registro agregado, si está en modo lectura o escritura, el número y el valor del registro.

Para eliminar un registro previamente configurado, debe seleccionarse de la Lista de Registros y dando clic derecho sobre el mismo, seleccione la opción Eliminar registro en el menú contextual desplegado. Para eliminar todos los registros configurados de la Lista de Registros, presione el botón Clear, Simbus le solicitará confirmar esta acción.

Después de configurar la Lista de Registros a consultar, en el panel de Dispositivo de Simbus, se selecciona la Slave Id a consultar (Identificación de esclavo), el nombre del dispositivo y el intervalo entre consultas, dado en milisegundos. Si se desea realizar una sola consulta, el valor -1 debe ser asignado al intervalo entre consultas.
Por último solo debe presionar el botón Add, para agregar el dispositivo configurado a la Lista de Dispositivos.

Haciendo doble clic sobre el dispositivo seleccionado, en el panel Detalle Consulta, se visualizan
los registros configurados. Para iniciar la simulación despliegue el menú contextual
sobre un dispositivo y seleccione la opción Iniciar simulación. Al iniciar la simulación se despliega la ventana Log Simulación, que informa los sucesos relacionados a esa simulación.

Para detener la simulación despliegue el menú contextual sobre un dispositivo y seleccione la opción Detener simulación. Puede exportar el registro de sucesos a través del botón Exportar en la ventana Log Simulación.

Master RTU/ASCII

En este caso Simbus habilita el panel de Dispositivo serial RTU/ASCII. Seleccione el puerto, y configure los parámetros Velocidad del puerto, Bits de datos, Bits de parada y Paridad. Los registros se seleccionan y se eliminan de la misma forma que en la caso de Master TCP. También de manera análoga al caso Master TCP se agrega el dispositivo a la Lista de Dispositivos, mediante el botón Add en el Panel de Dispositivo.

Haciendo doble clic sobre el dispositivo seleccionado, en el panel Detalle de consulta, se visualizan los registros configurados. Para iniciar la simulación despliegue el menú contextual sobre un dispositivo y seleccione la opción Iniciar simulación. Al iniciar la simulación la ventana de Log simulación aparecerá de igual forma que en el caso Master TCP.

Al configurar una nueva simulación en modo Modbus Master, Simbus le permite seleccionar el tipo de comunicación y el tiempo de espera máximo, en el panel Comunicación. Es necesario dar clic al botón Apply para ajustar la vista general con la configuración seleccionada.

Slave TCP

En este caso Simbus habilita el panel de dispositivo TCP. Seleccione la dirección IP y el número de puerto asignado al dispositivo Slave. Haciendo clic en el botón Lista de Registros del panel de Dispositivo visualice el diálogo que permite configurar los registros a consultar. Haciendo clic derecho sobre el diálogo de registros configurados, despliegue el menú contextual y seleccione la acción añadir registro.

Seleccione el Tipo de registro a consultar, el número de registro y si lo requiere active la opción de escribir, dando un valor o un rango para generar el valor a

escribir aleatoriamente. Al dar clic en OK, inmediatamente el registro es agregado en la Lista de Registros. Esta muestra el tipo de registro agregado, si está en modo lectura o escritura, el número y el valor del registro.

Para eliminar un registro previamente configurado, debe seleccionarse de la Lista de Registros y dando clic derecho sobre el mismo, seleccione la opción Eliminar registro en el menú contextual desplegado. Para eliminar todos los registros configurados de la Lista de Registros, presione el botón Clear, Simbus le solicitará confirmar esta acción.

Después de configurar la Lista de Registros que escuchará consultas, en el panel de Dispositivo de Simbus, se selecciona la Slave Id (Identificación de esclavo) y el nombre del dispositivo. Por último solo debe presionar el botón Add, para agregar el dispositivo configurado a la Lista de Dispositivos.

Haciendo doble clic sobre el dispositivo seleccionado, en el panel Detalle Consulta, se visualizan los registros configurados. Para iniciar la simulación despliegue el menú contextual sobre un dispositivo y seleccione la opción Iniciar simulación. Al iniciar la simulación se despliega la ventana Log Simulación, que informa los sucesos relacionados a esa simulación.

Para detener la simulación despliegue el menú contextual sobre un dispositivo y seleccione la opción Detener simulación. Puede exportar el registro de sucesos a través del botón Exportar en la ventana Log Simulación.

Slave RTU/ASCII

En este caso Simbus habilita el panel de Dispositivo serial RTU/ASCII. Seleccione el puerto, y configure
los parámetros Velocidad del puerto, Bits de datos, Bits de parada y Paridad.
Los registros se seleccionan y se eliminan de la misma forma que en la caso de Slave TCP.
También de manera análoga al caso Master TCP se agrega el dispositivo a la Lista de Dispositivos.
Mediante el botón Add en el Panel de Dispositivo.

ANEXO 4: DIAGRAMAS DE CLASE

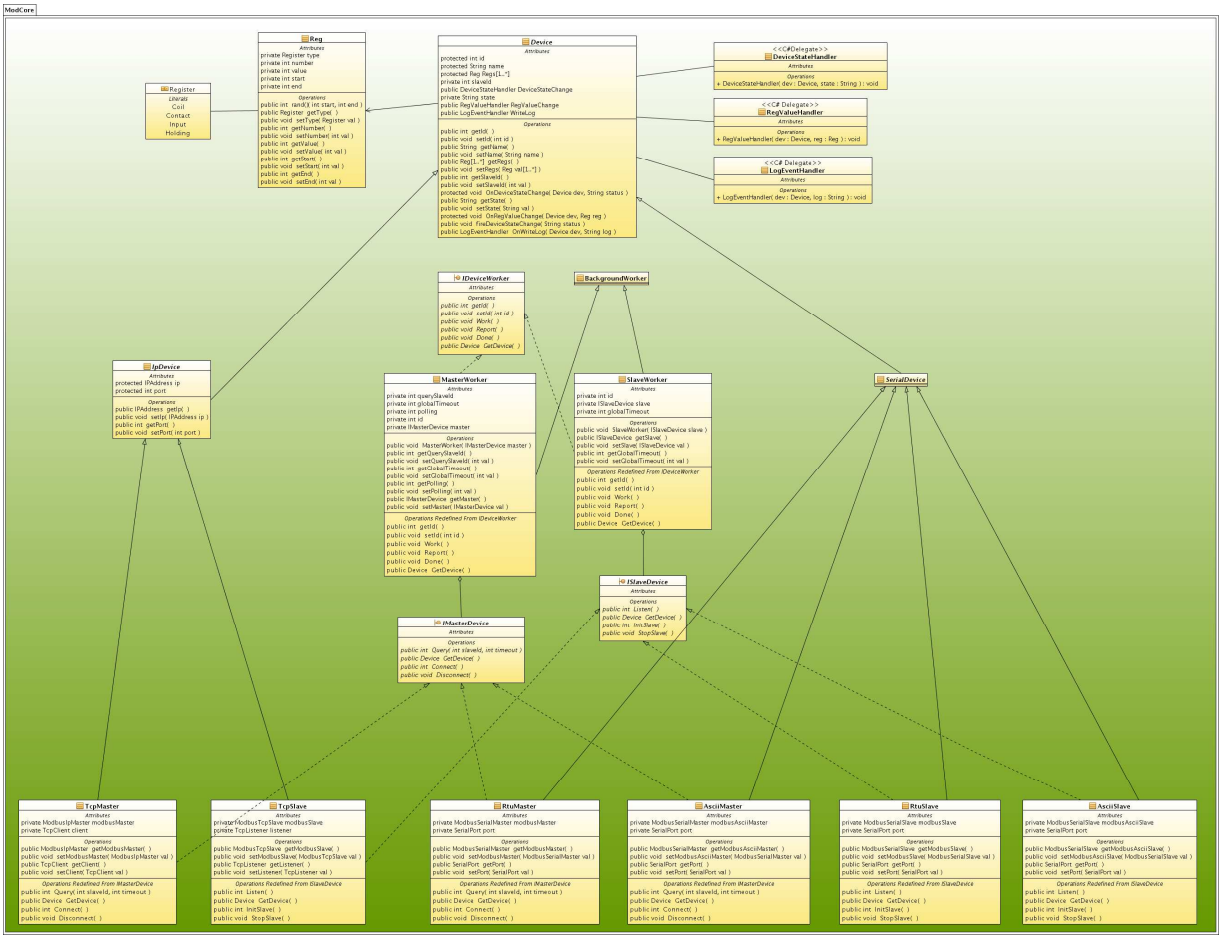


DIAGRAMA DE CLASES MODCORE

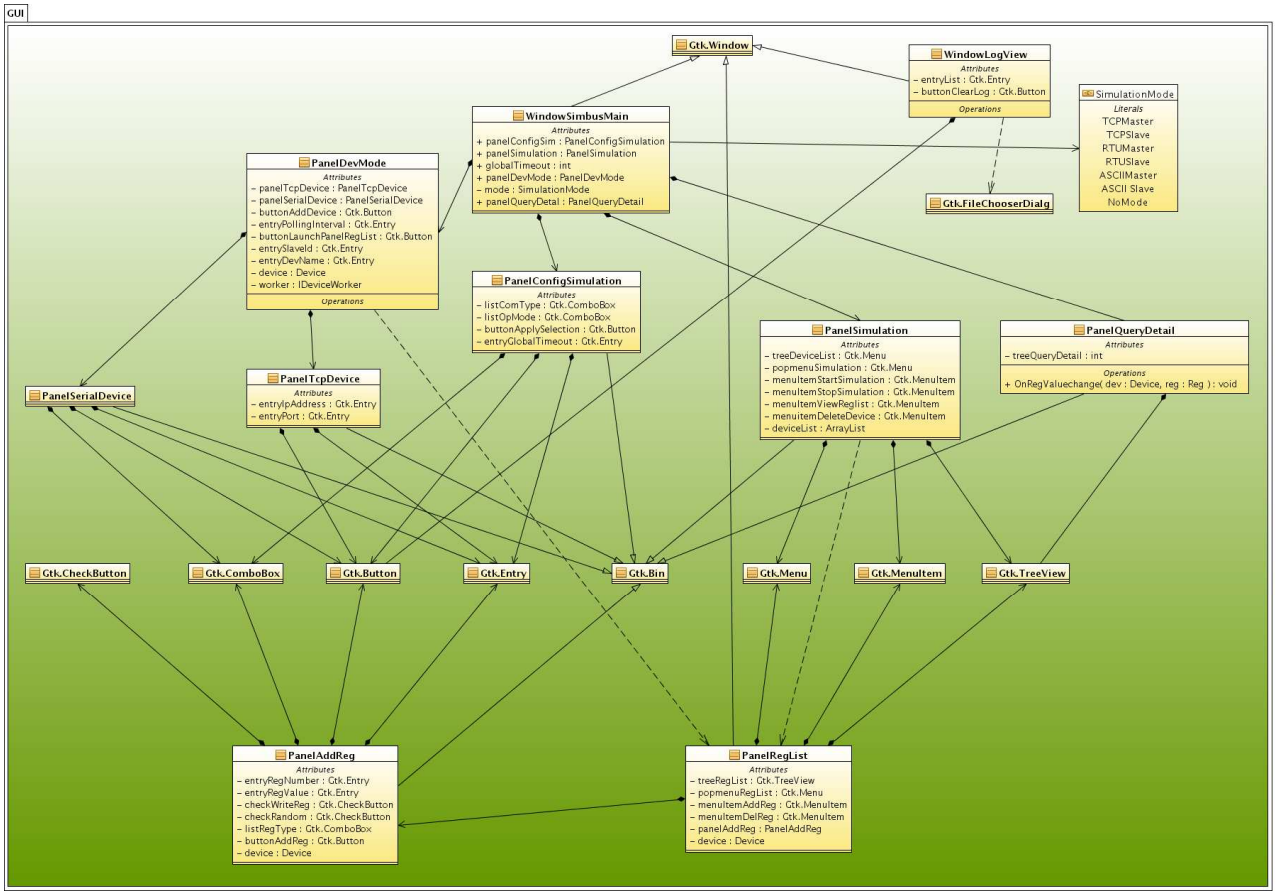
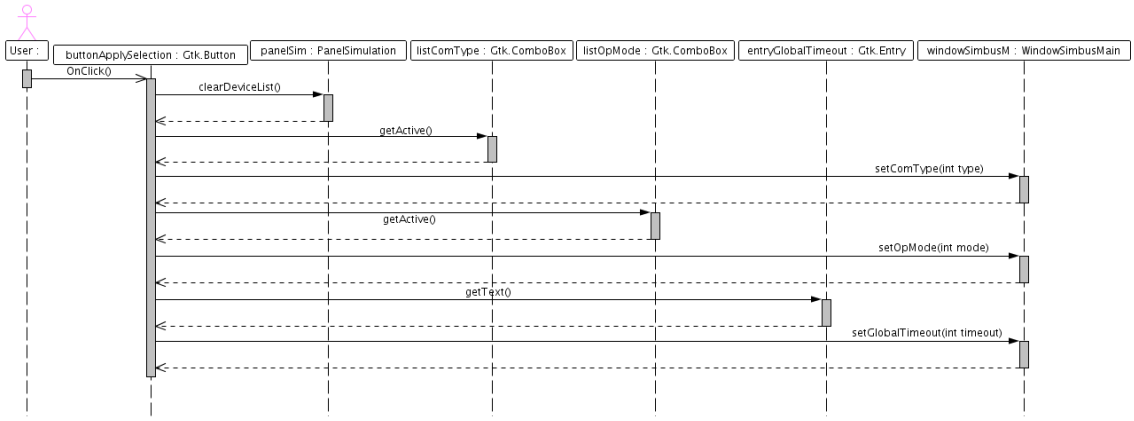
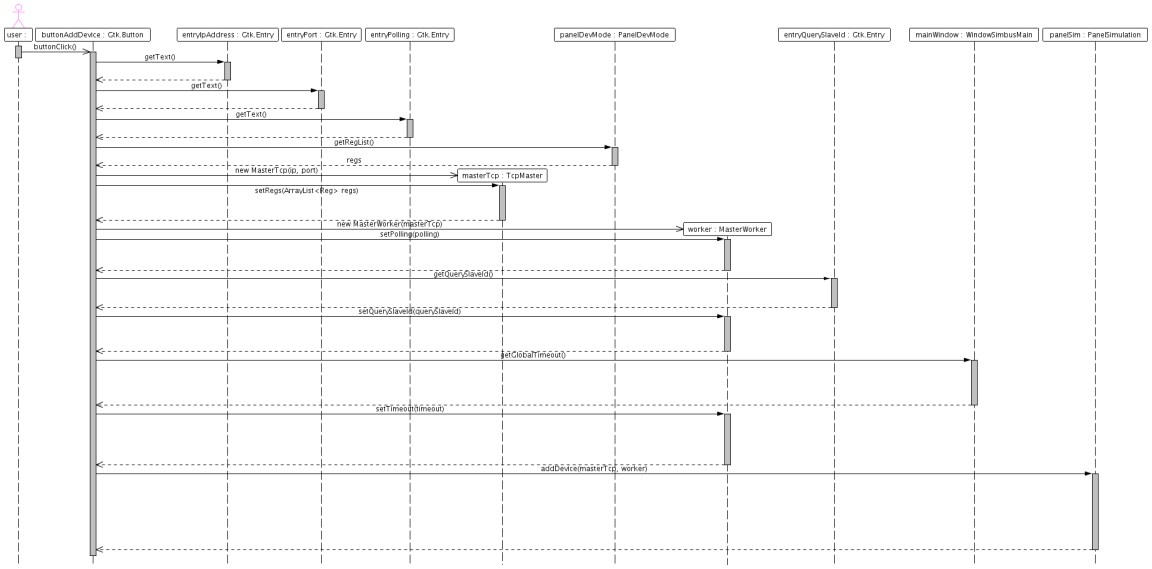


DIAGRAMA DE CLASES GUI

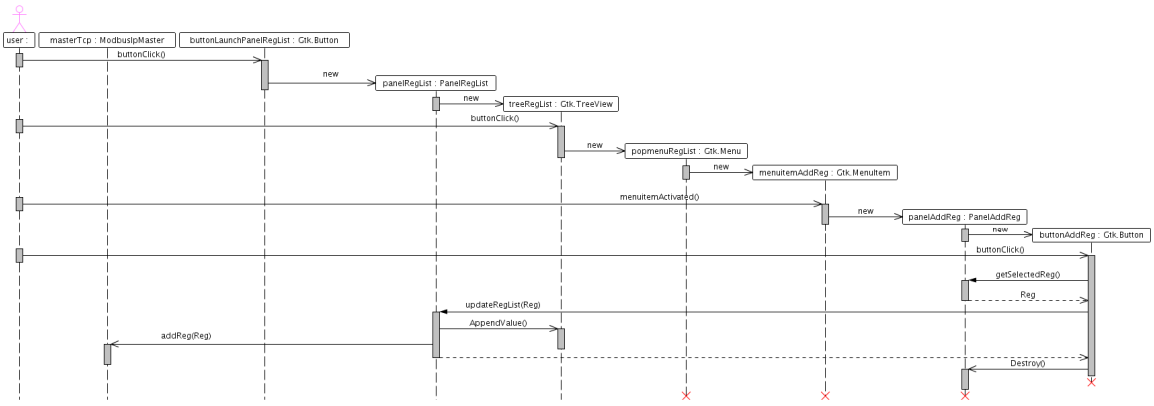
ANEXO 5: DIAGRAMAS DE SECUENCIA



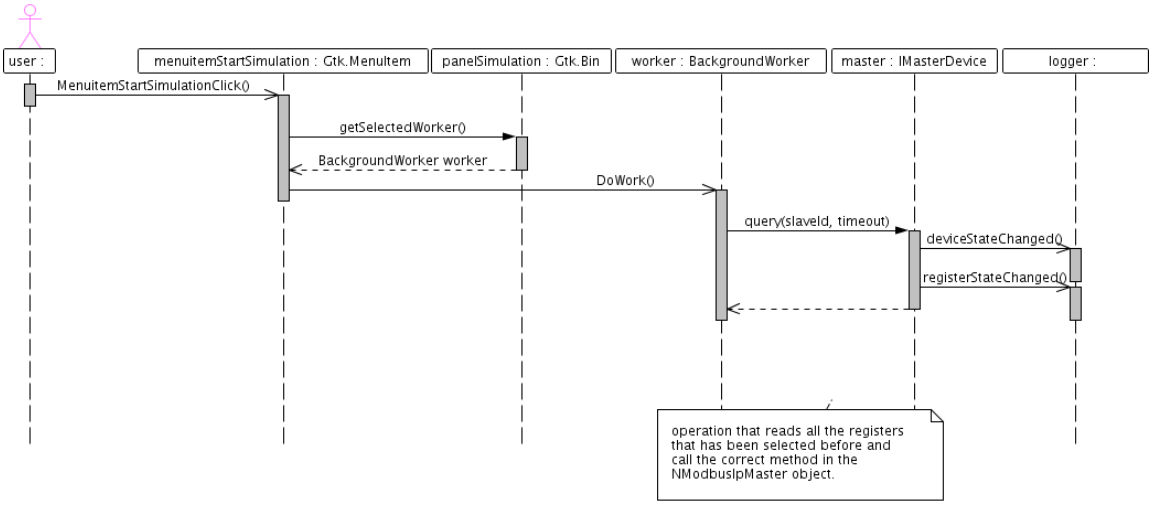
MFS01_FS01_F01_F02



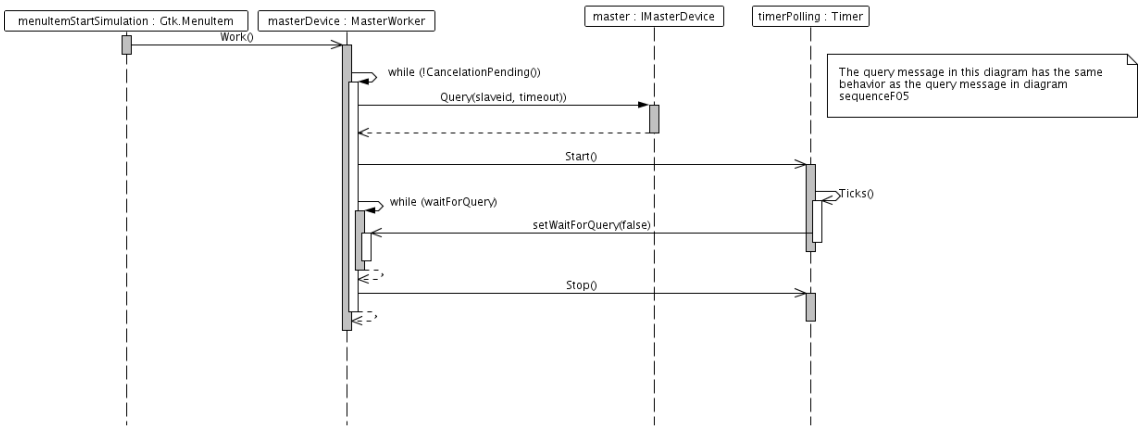
MFS01_FS01_F03



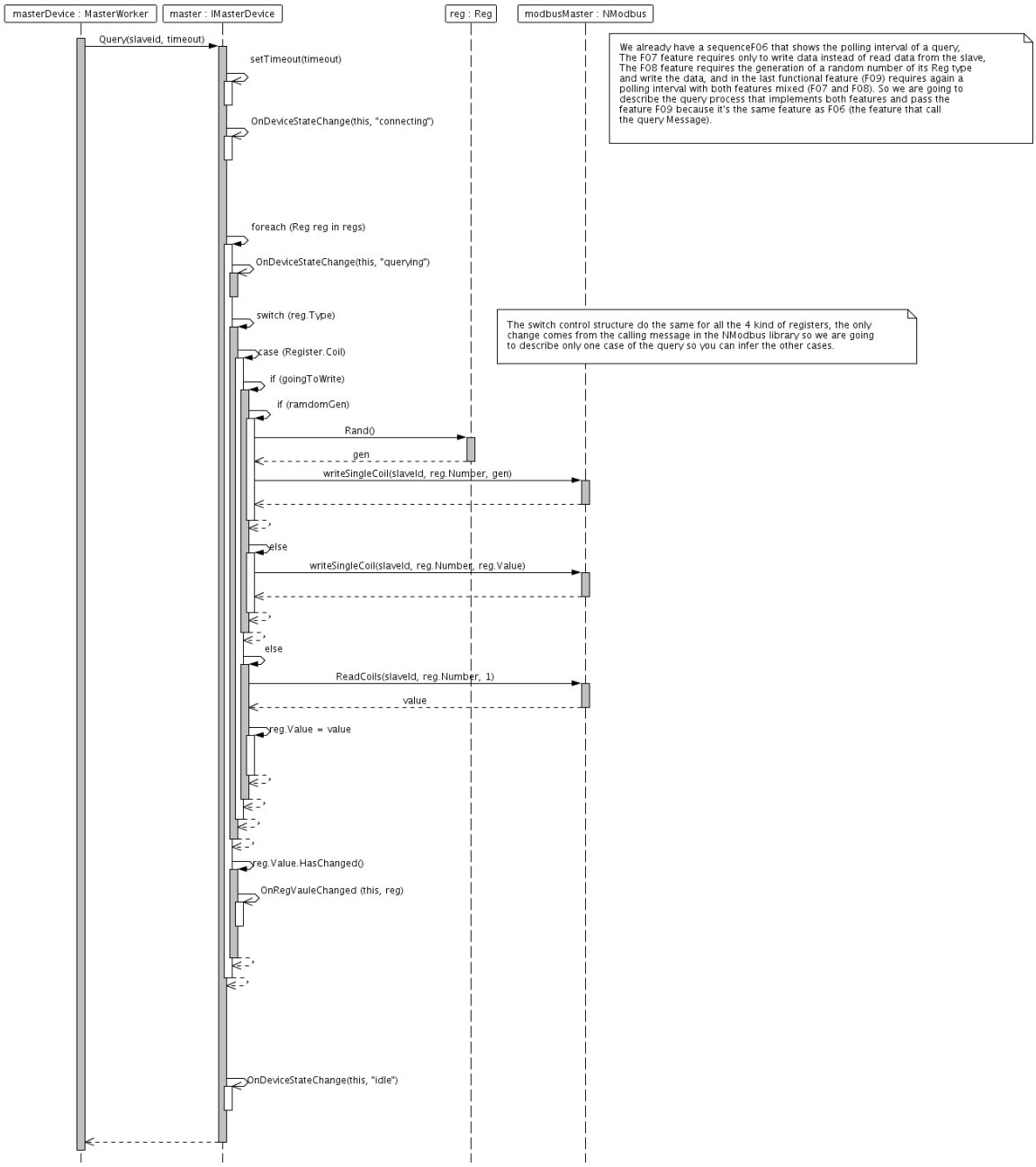
MFS01_FS01_F04



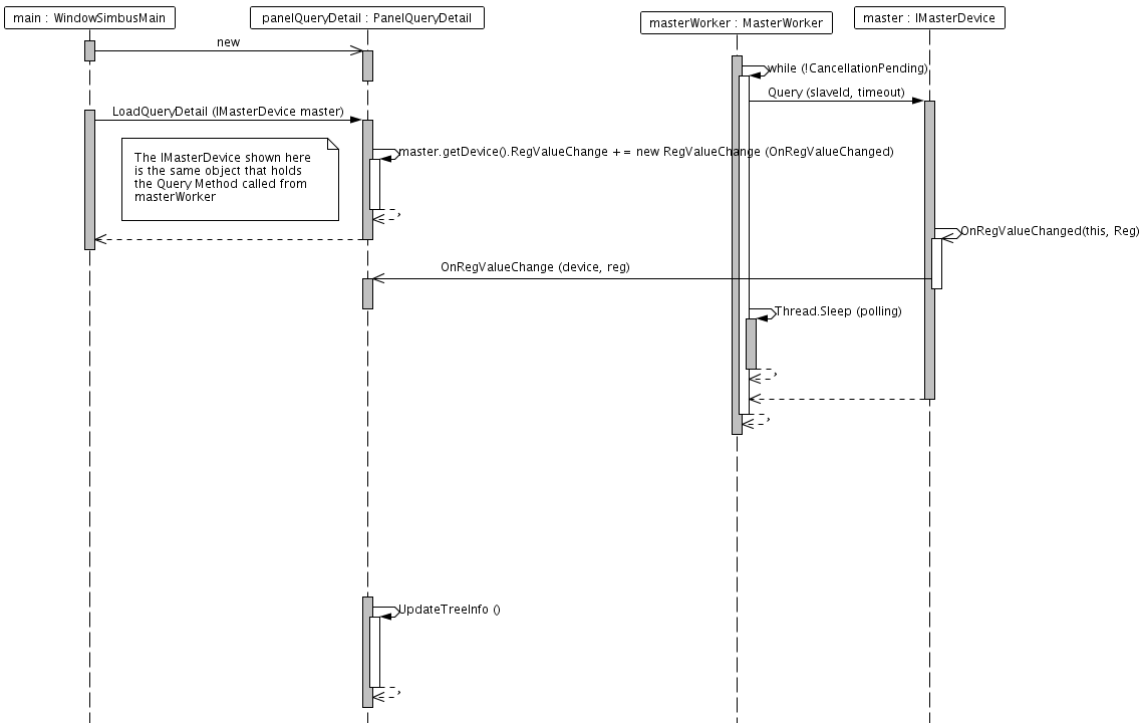
MFS01_FS01_F05



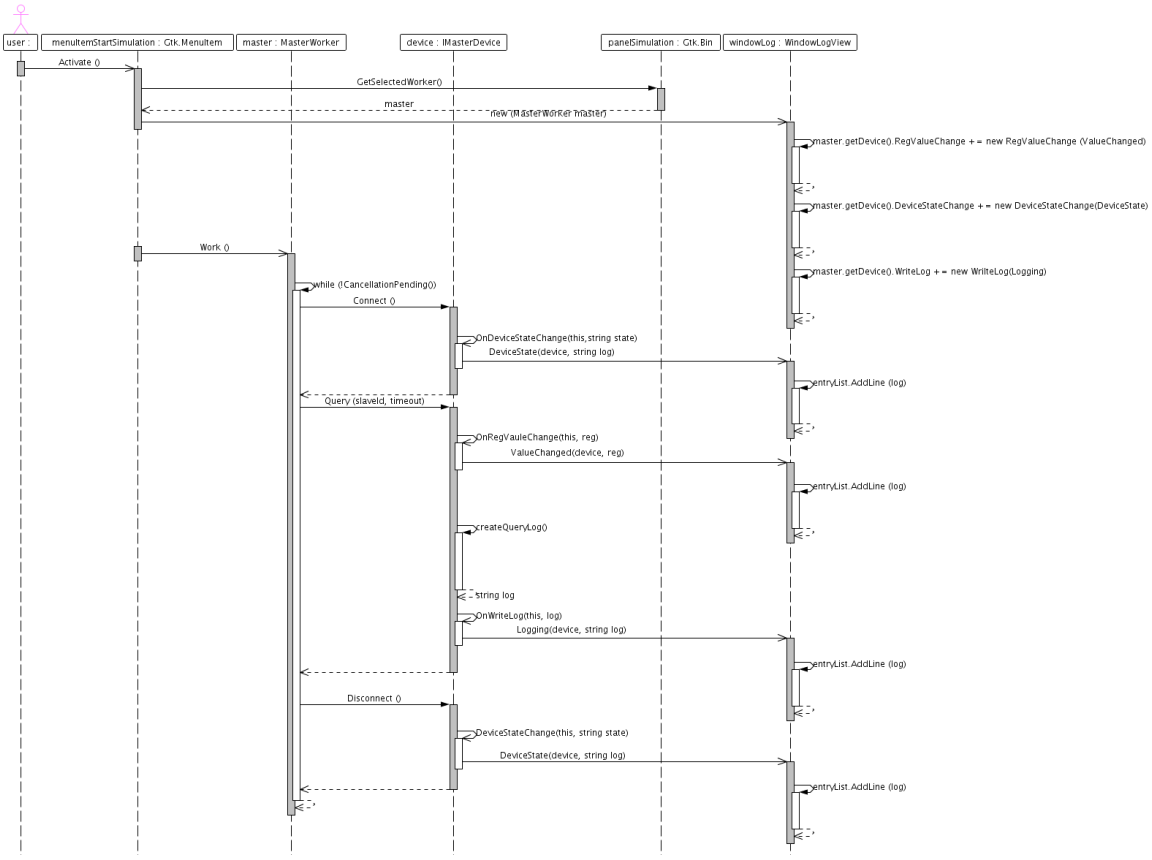
MFS01_FS01_F06



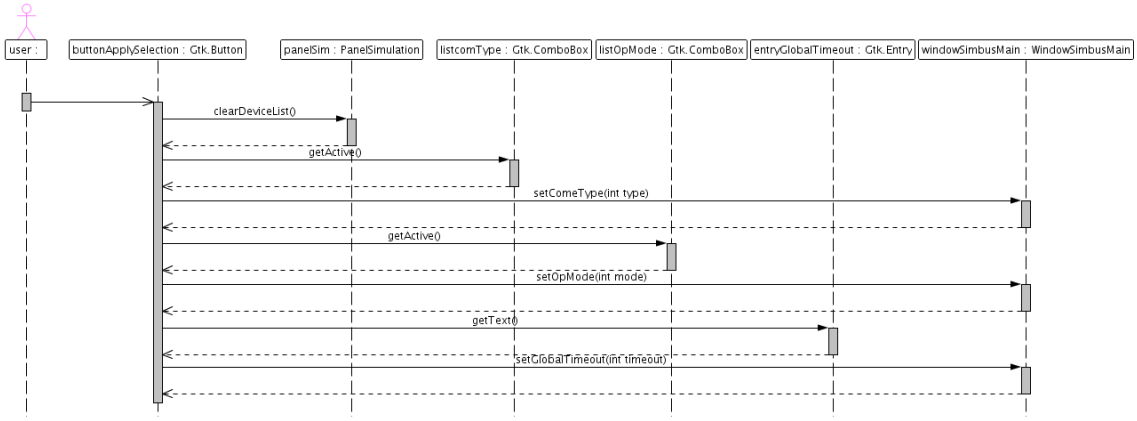
MFS01_FS01_F07_F08_F09



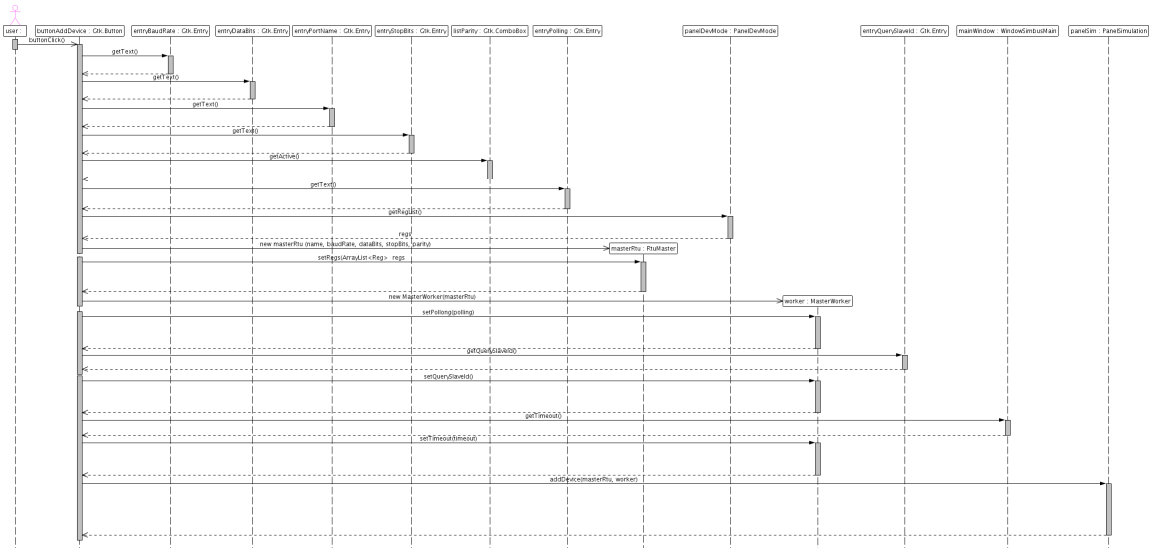
MFS01_FS01_F10



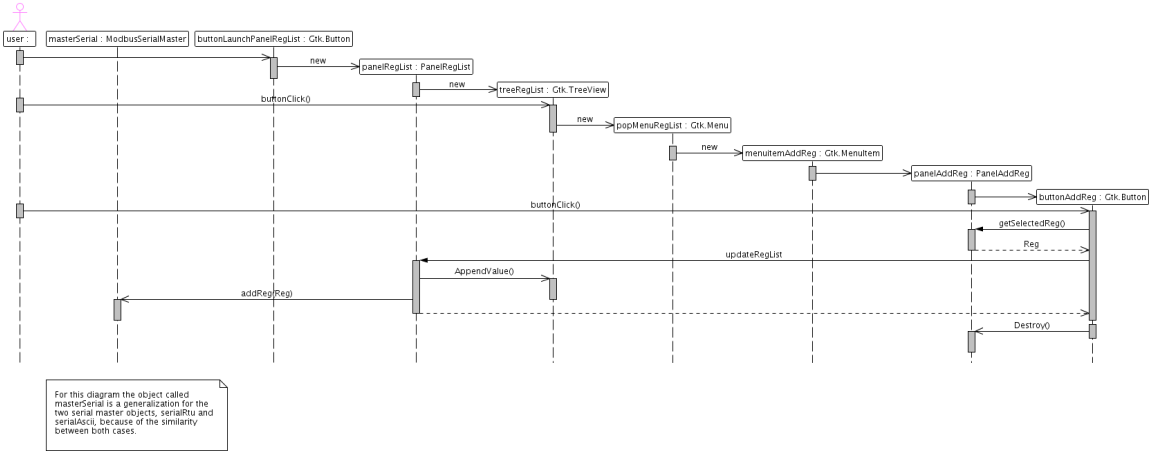
MFS01_FS01_F11



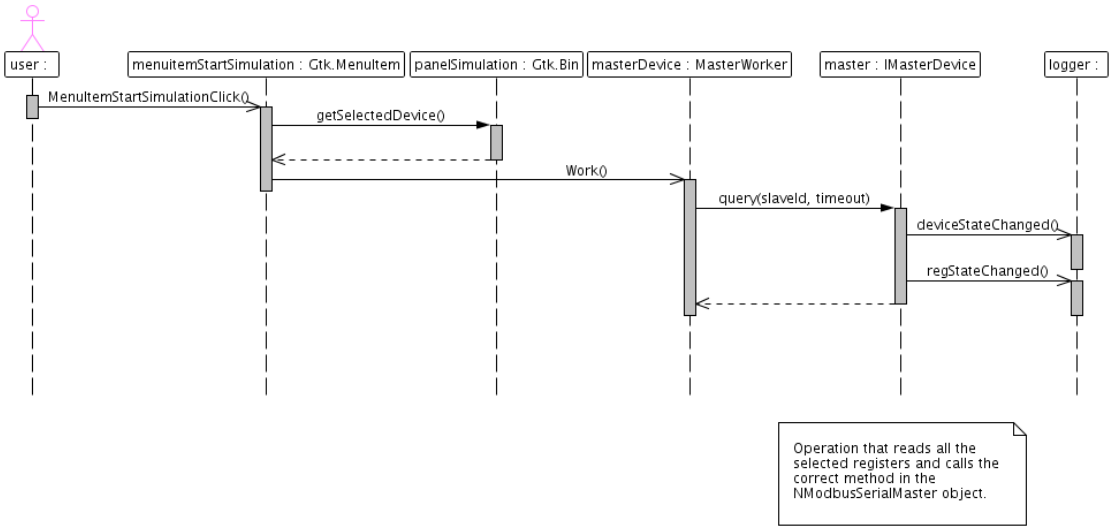
MFS01_FS02_F01_F02



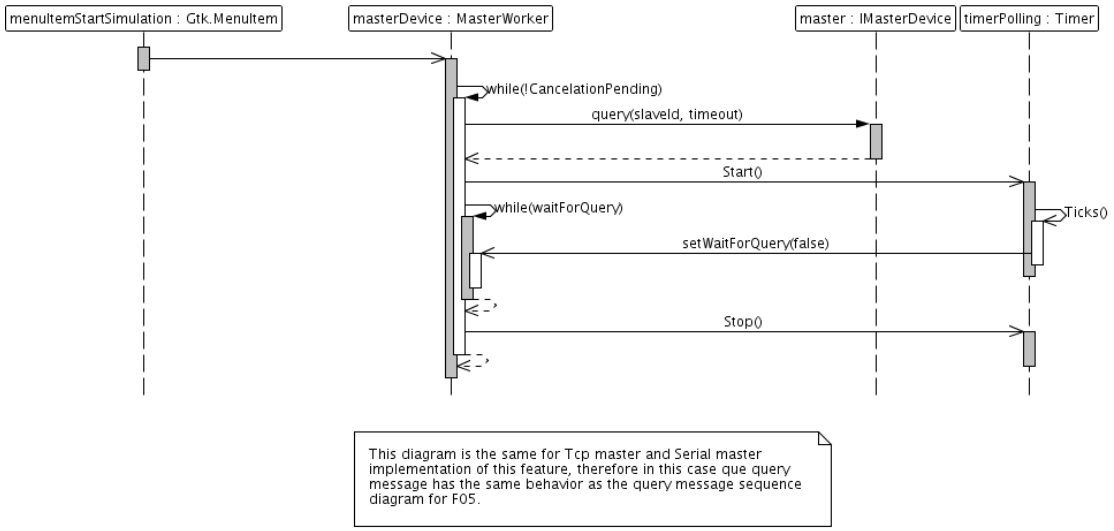
MFS01_FS02_F03



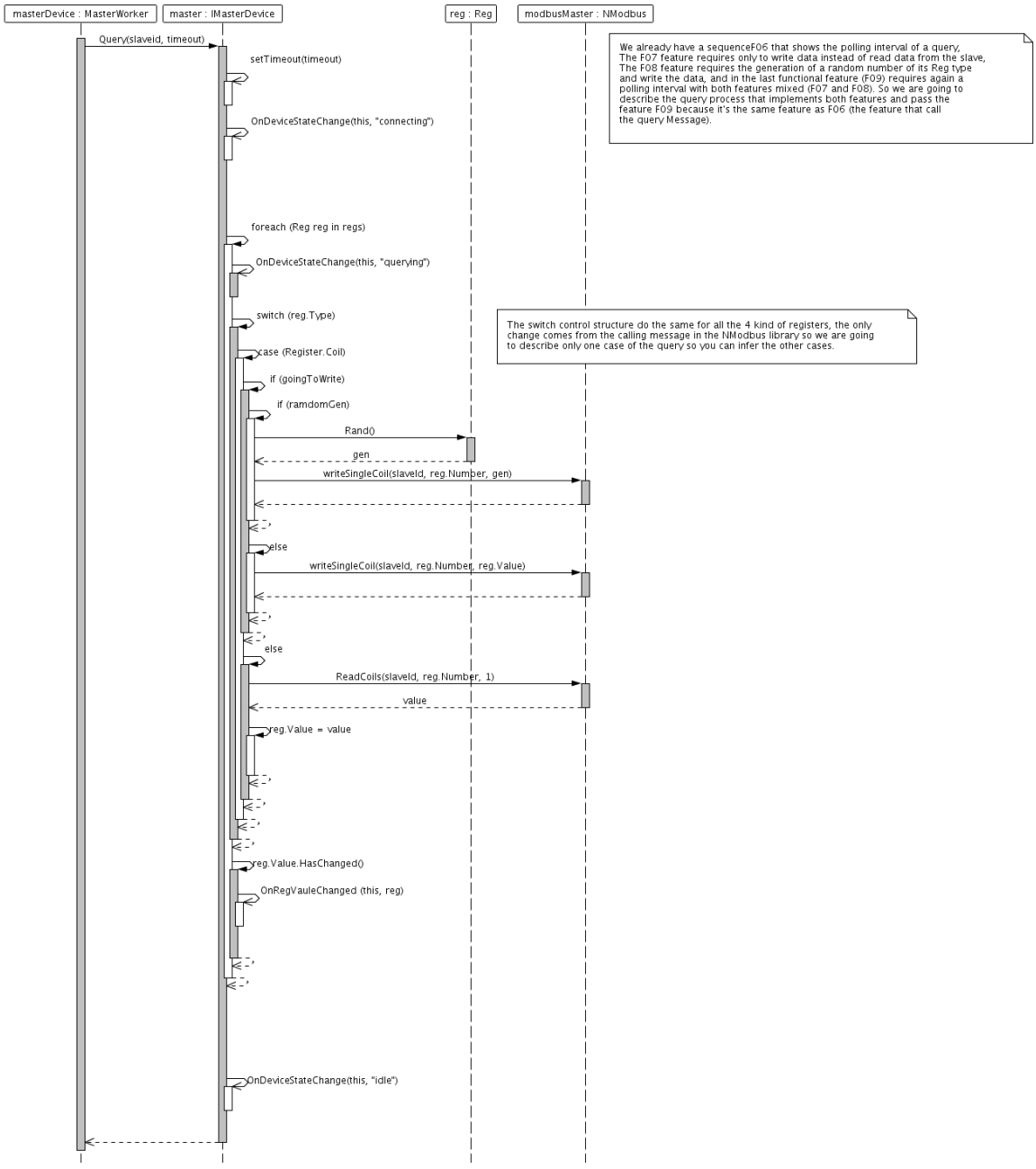
MFS01_FS02_F04



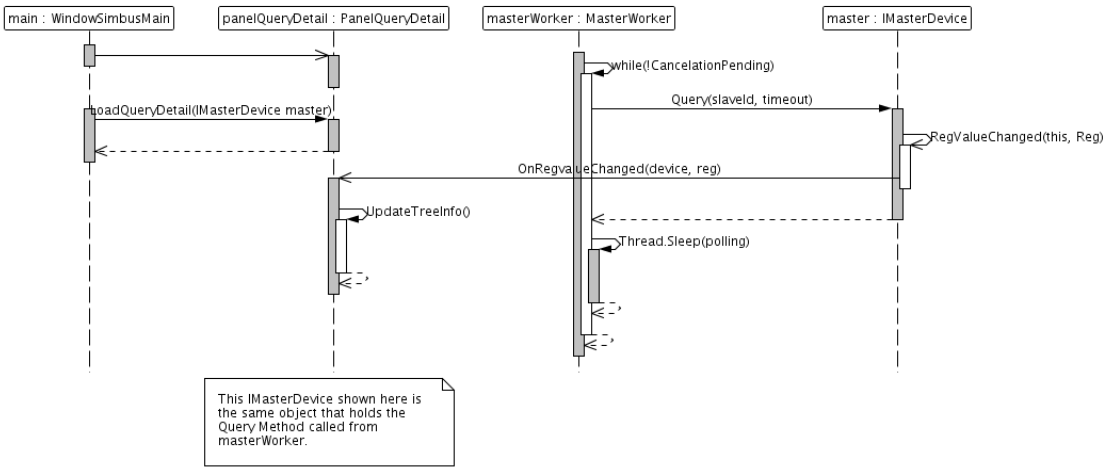
MFS01_FS02_F05



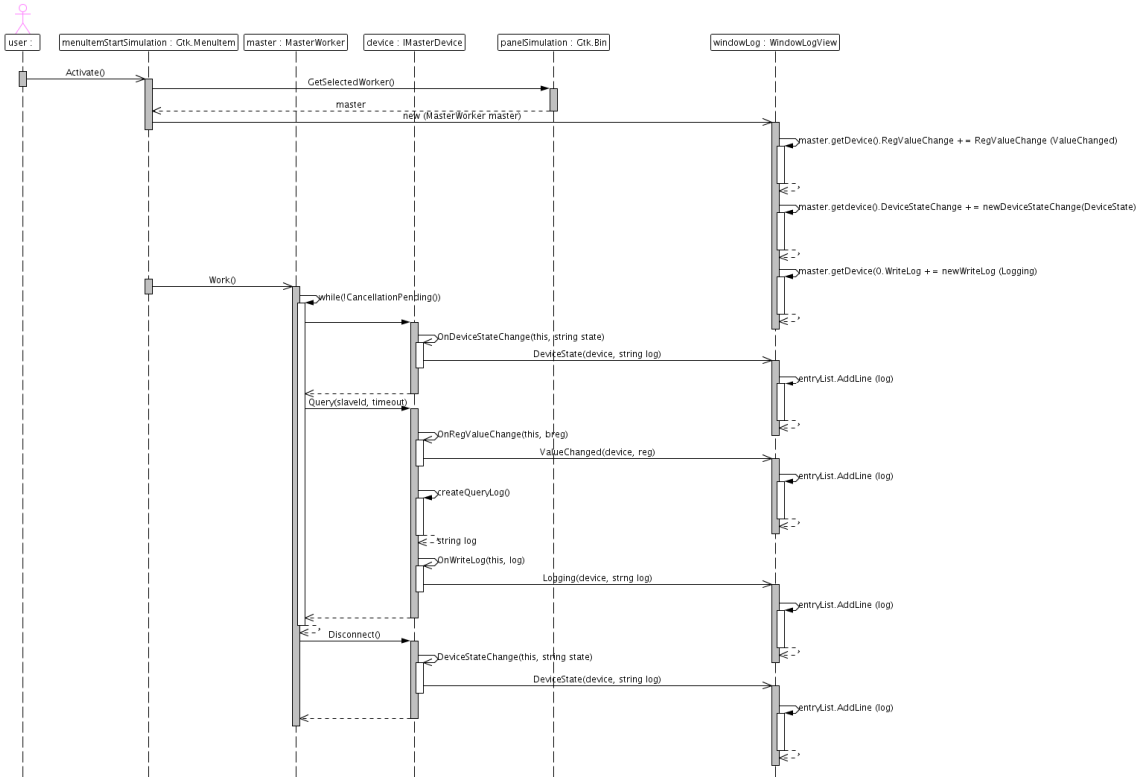
MFS01_FS02_F06



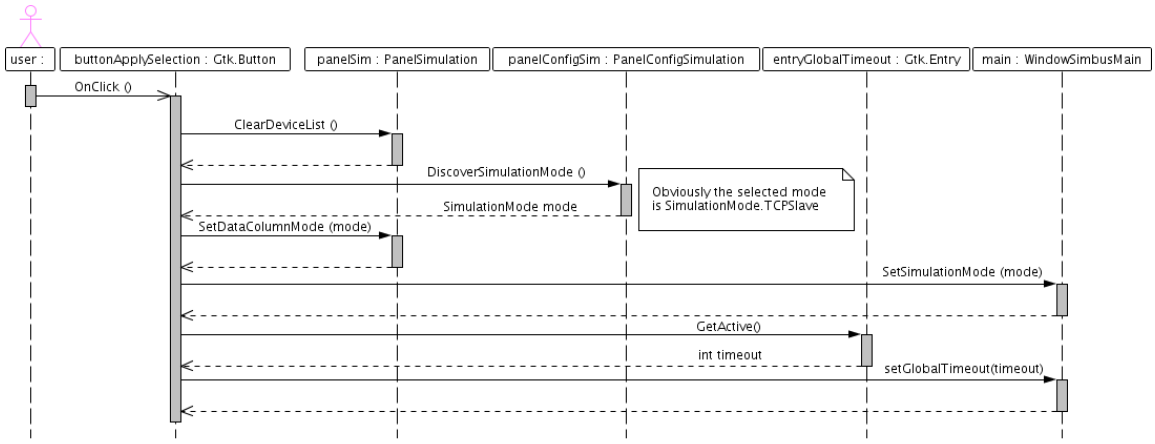
MFS01_FS02_F07_F08_F09



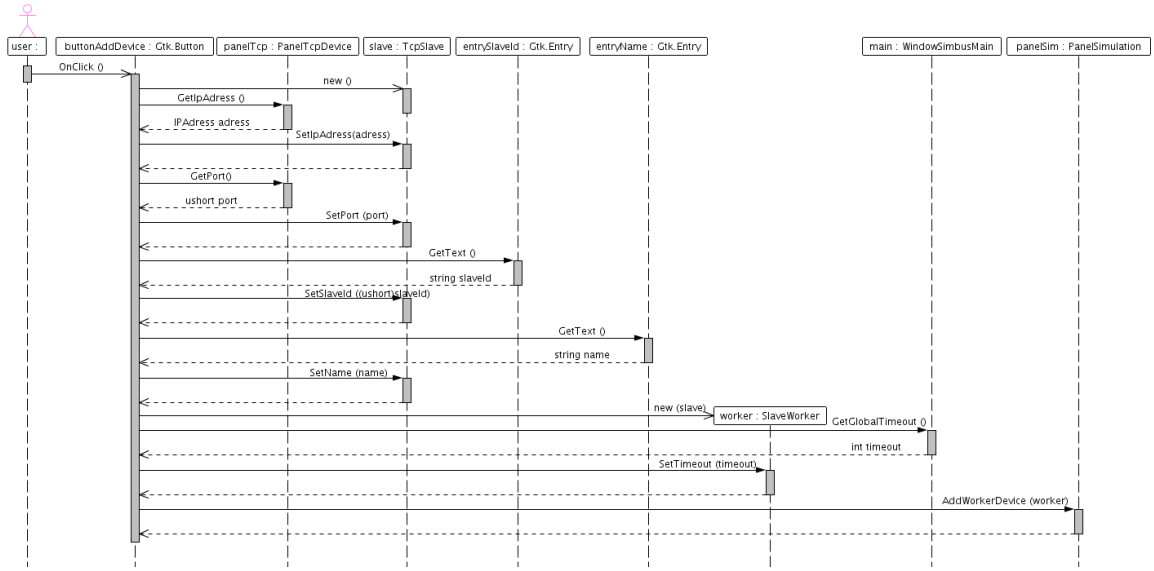
MFS01_FS02_F10



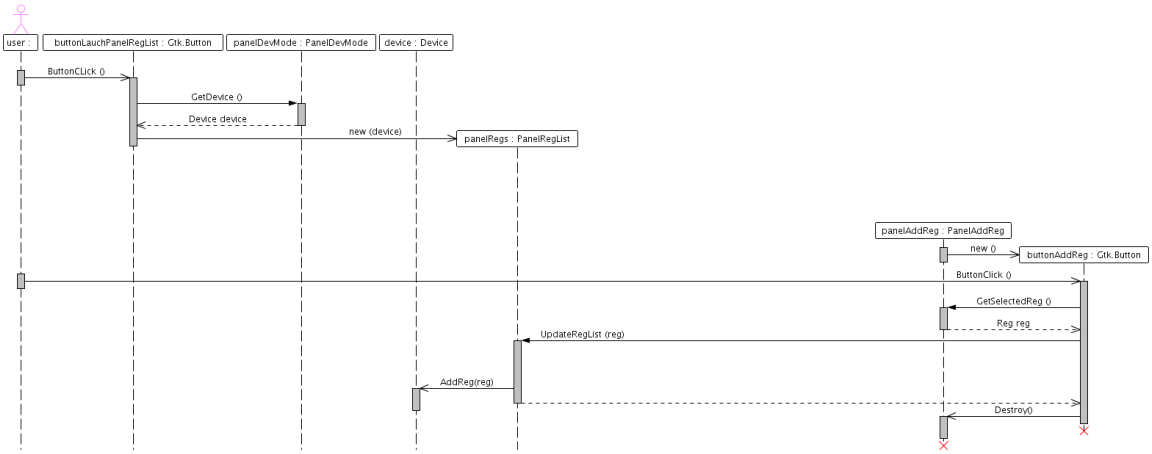
MFS01_FS02_F11



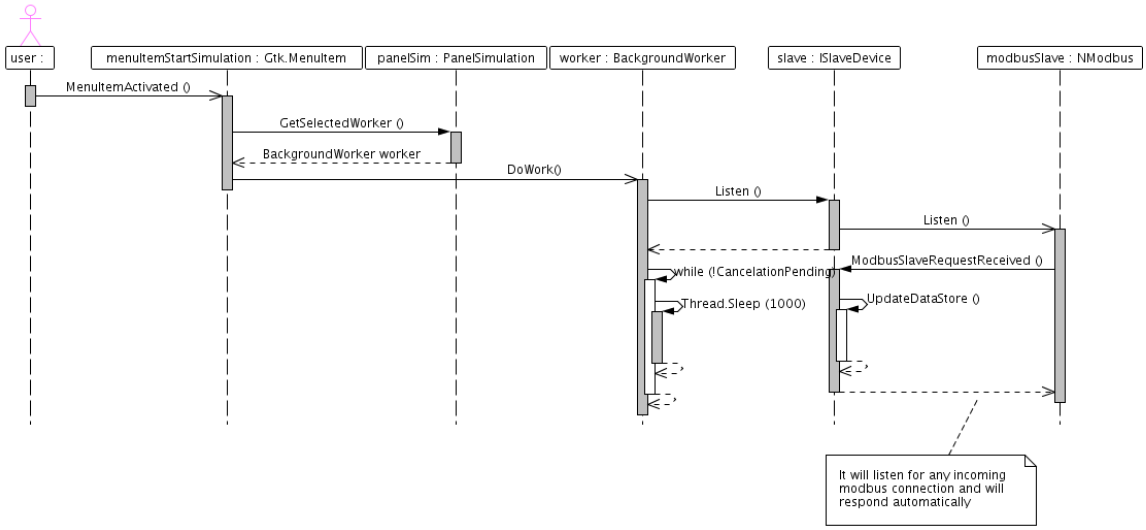
MFS02_FS01_F01_F02



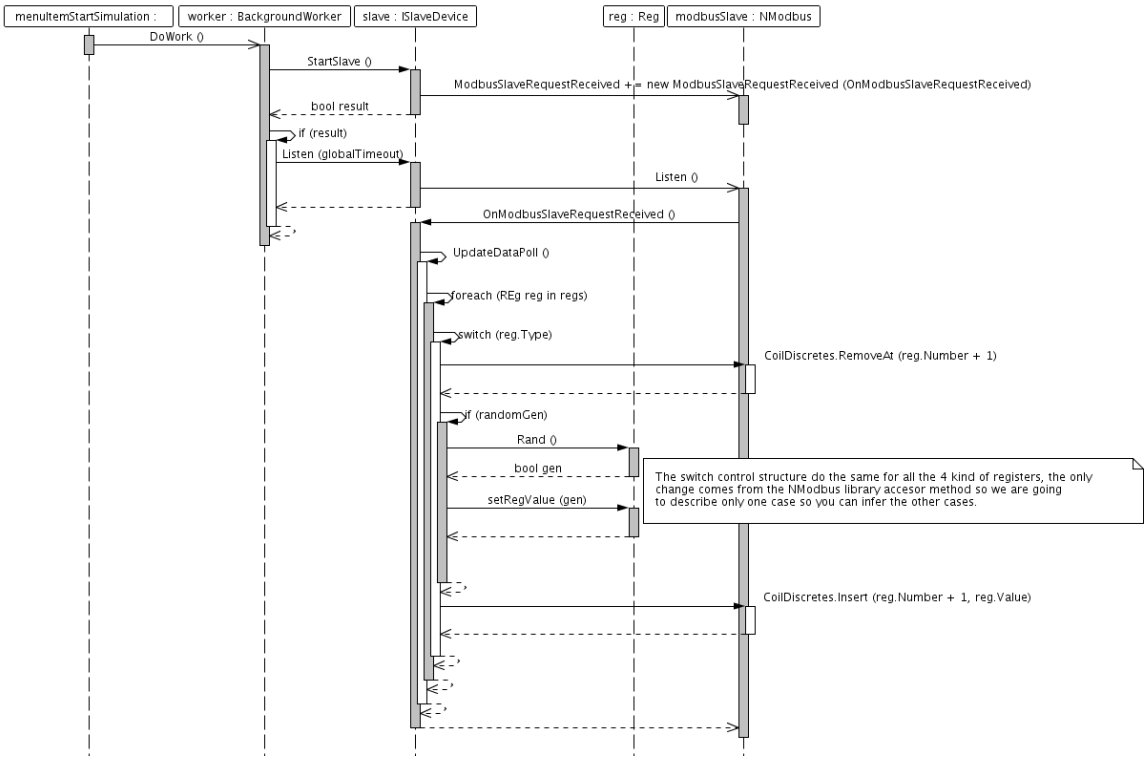
MFS02_FS01_F03



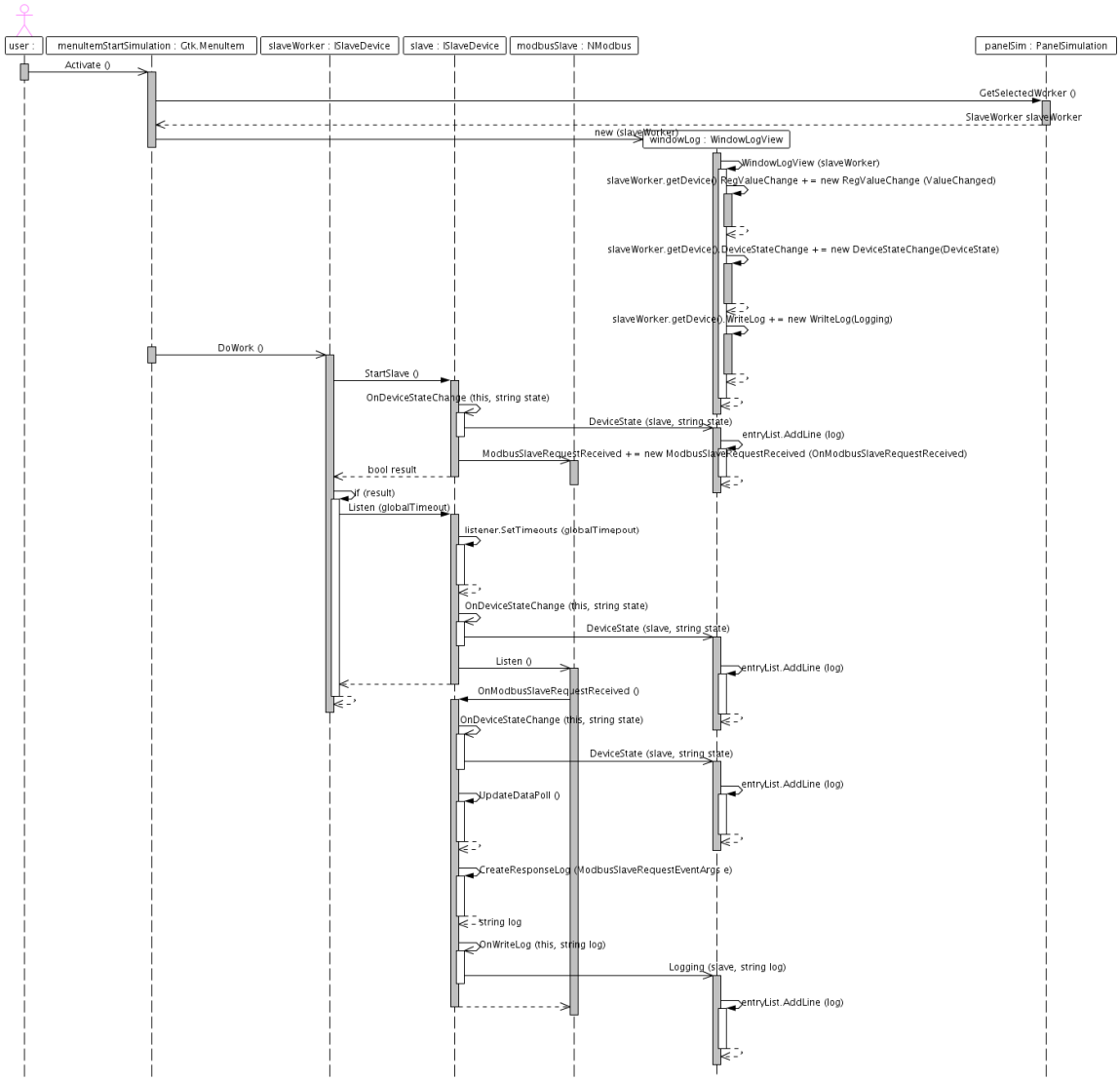
MFS02_FS01_F04



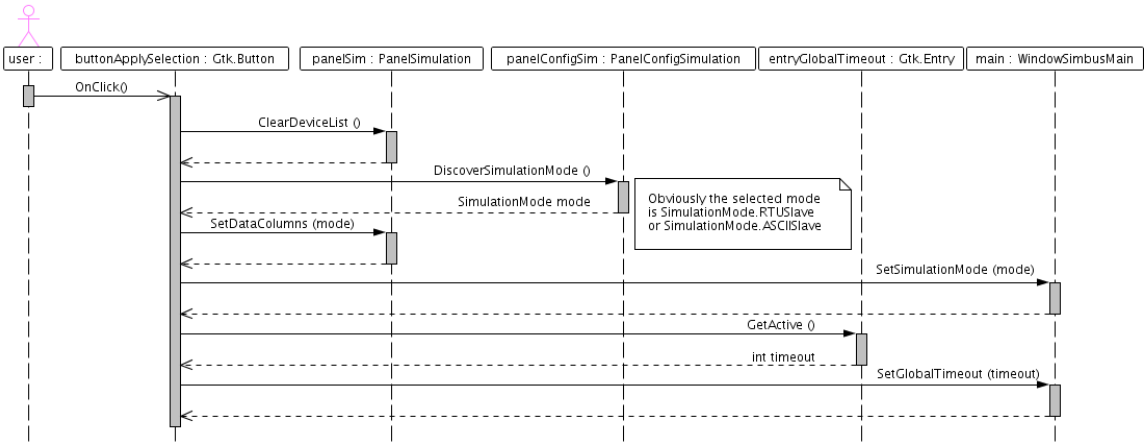
MFS02_FS01_F05



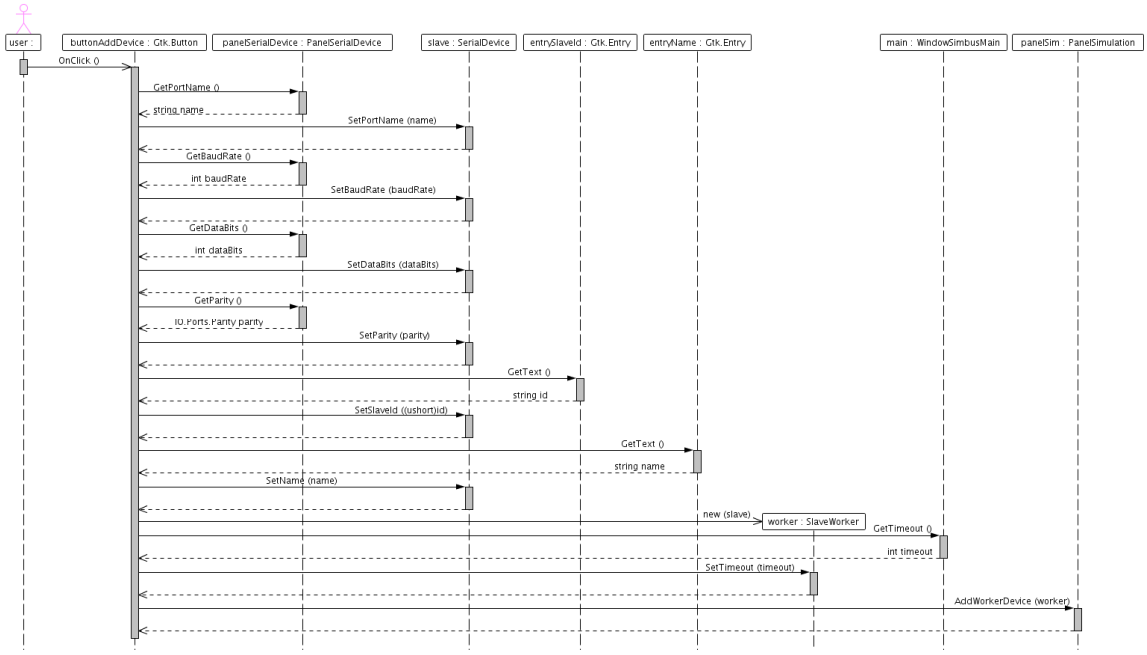
MFS02_FS01_F06



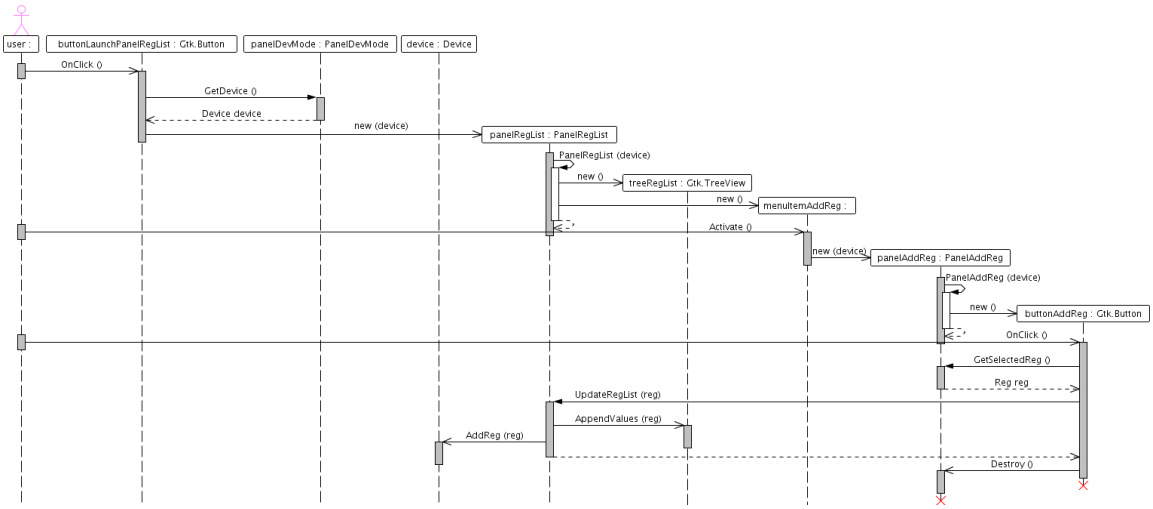
MFS02_FS01_F07



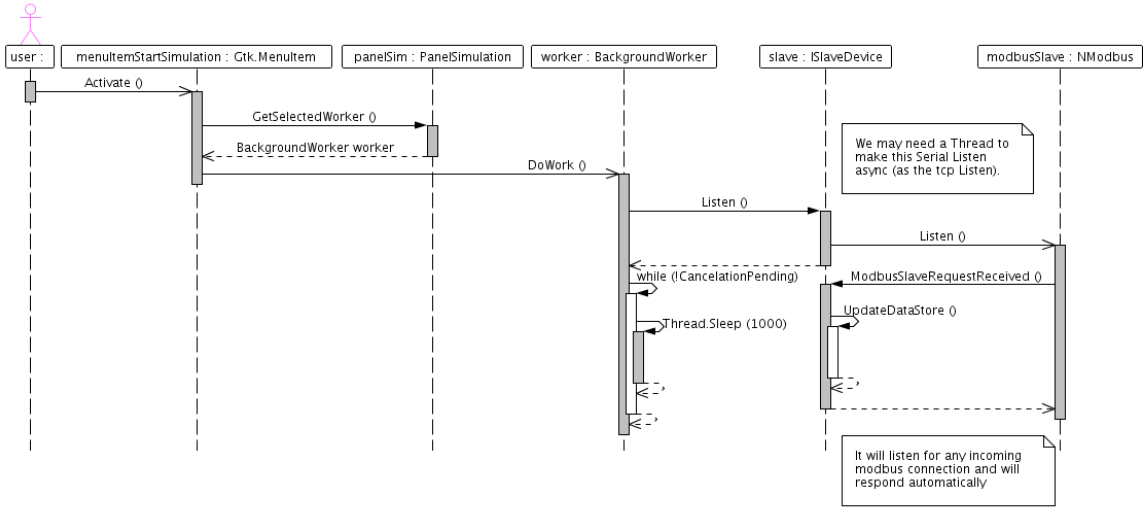
MFS02_FS02_F01_F02



MFS02_FS02_F03



MFS02_FS02_F04



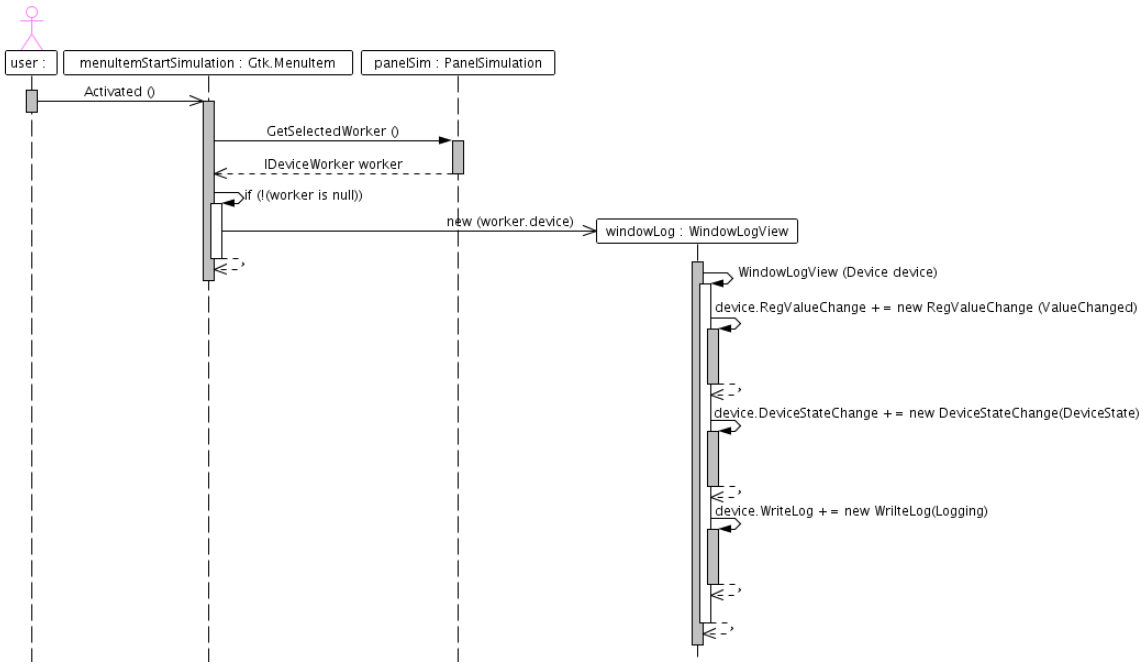
MFS02_FS02_F05

This Diagram doesn't exist because is exactly the same diagram showed in the sequenceMFS02_FS01_F06 as the desing works with a ISlaveDevice it doesn't matter if the slave is Rtu, Ascii or Tcp, it behaves equally.

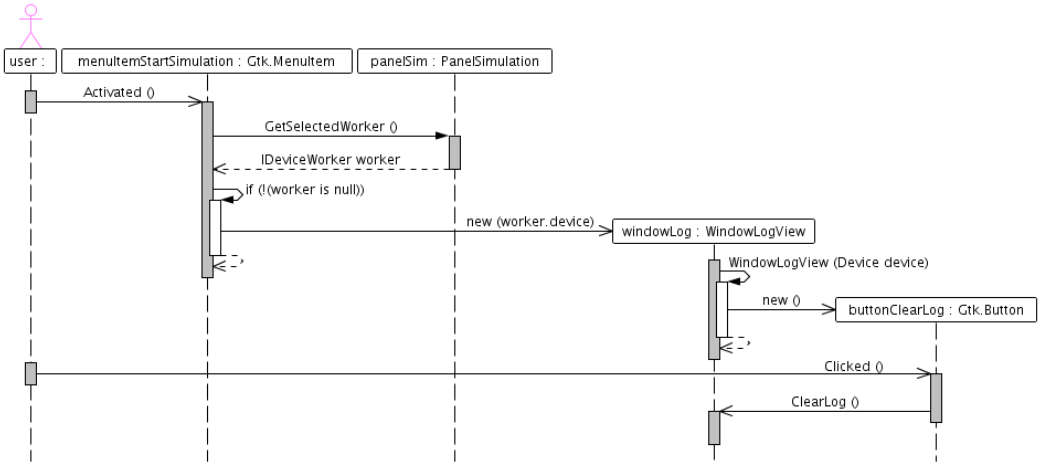
MFS02_FS02_F06

Again this Diagram doesn't exist because is exactly the same diagram showed in the sequenceMFS02_FS01_F07 as the desing works with a ISlaveDevice and a SlaveWorker it doesn't matter if the slave is Rtu, Ascii or Tcp, it behaves equally.

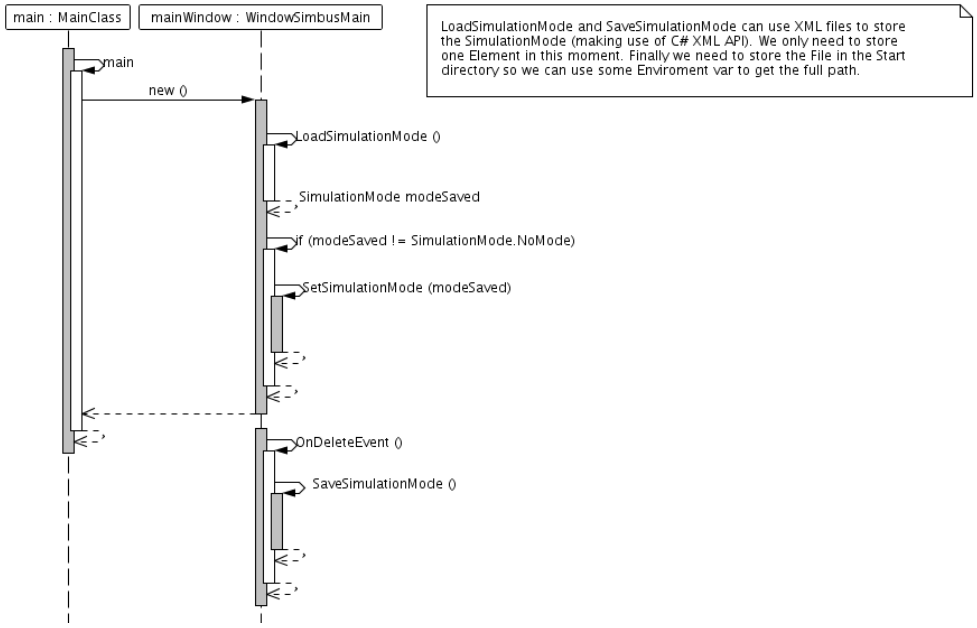
MFS02_FS02_F07



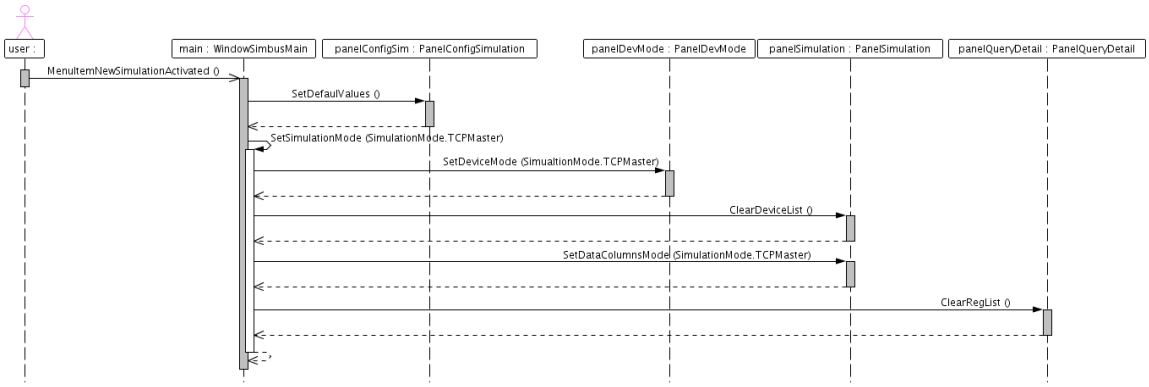
MFS03_FS01_F01



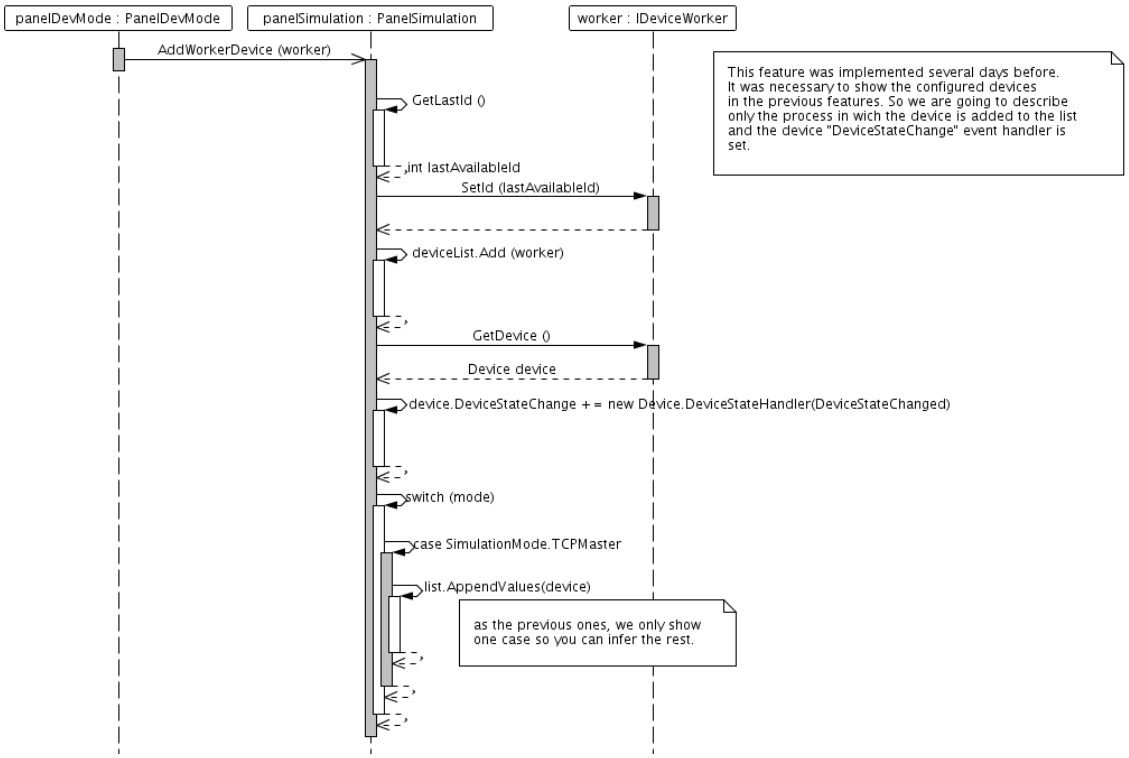
MFS02_FS02_F03



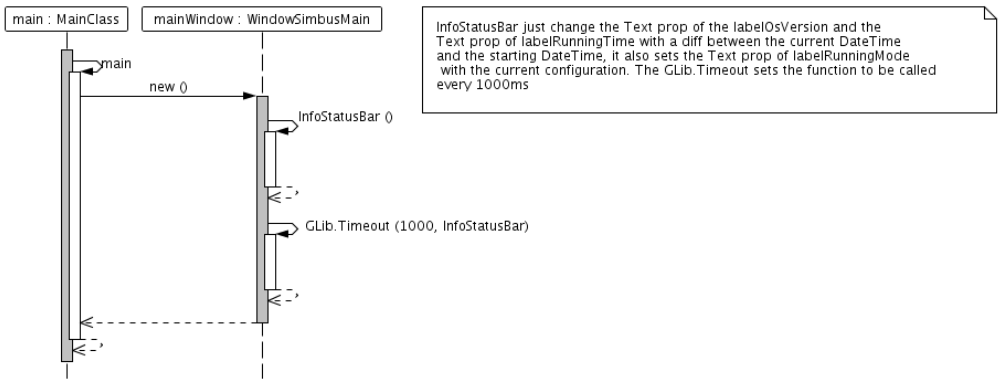
MFS03_FS01_F04



MFS03_FS01_F05



MFS03_FS01_F06



MFS03_FS01_F07