UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

PROPUESTA DE MODULARIZACIÓN DE SOFTWARE SCADA PARA MICRORRED IMPLEMENTADA EN LA COMUNIDAD DE HUATACONDO UTILIZANDO LENGUAJE C#

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

RAÚL ANDRÉS UGARTE MUÑOZ

PROFESOR GUÍA:
PATRICIO MENDOZA ARAYA

MIEMBROS DE LA COMISIÓN:
FERNANDO LANAS MONTECINOS
ANDRÉS CABA RUTTE

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: RAÚL ANDRÉS UGARTE MUÑOZ
FECHA: 2019
PROF. GUÍA: PATRICIO MENDOZA ARAYA

PROPUESTA DE MODULARIZACIÓN DE SOFTWARE SCADA PARA MICRORRED IMPLEMENTADA EN LA COMUNIDAD DE HUATACONDO UTILIZANDO LENGUAJE C#

Actualmente las microrredes son la solución que se está investigando e implementando tanto para redes eléctricas aisladas como conectadas a la red, para abastecer de manera local las necesidades energéticas de las comunidades. En el año 2009, se implementó un proyecto de microrred para la comunidad de Huatacondo, ESUSCON (Energía Sustentable Condor). Este se encuentra en funcionamiento desde el año 2010 y ha sido asistido de diversas maneras por el equipo del Centro de Energía, de la Universidad de Chile.

En este contexto, el sistema que controla ESUSCON hace uso de un programa desarrollado por el equipo del Centro de Energía. A través de los años el programa ha sufrido diferentes cambios y actualmente no se conoce con claridad la interacción de los diferentes componentes en su operación, lo que hace difícil su modificación y mejora. Por esta razón, se propone abordar el programa y modularizarlo para mejorar su desempeño y separar sus partes según la función que cumplen. Particularmente, se propone modificar el archivo llamado *"form1"* y separar su estructura en el *Backend* y *Frontend* del SCADA.

La metodología utilizada para modularizar el código comienza abordándose a través del uso del diagrama de clases del sistema SCADA. Este permite entender la interacción entre las clases del sistema y proponer una estructura del programa que permita un funcionamiento coordinado entre módulos separados, con miras a generar menos fallas en el programa de control de la microrred en Huatacondo.

Para validar la metodología anterior se realizan pruebas al programa utilizando un servidor OPC local que simula las entradas para el SCADA. Para ello se simula la operación de la microrred recibiendo datos del servidor OPC en 4 pruebas durante 19 minutos cada una. Finalmente se utiliza la herramienta *"Performance Wizard"* del programa *"Microsoft Visual Studio 2010"* para hacer un perfil y analizar las métricas de desempeño del programa.

Luego de realizada la modularización y la prueba de funcionamiento en 4 casos, se pudo apreciar en los resultados del *"Performance Wizard"* que luego de la modularización se crean dos procesos *form*. Esto significa, que el proceso se divide en dos procesos, que ocupan alrededor de 35 % de la CPU cada uno, versus los casos sin modularizar con un solo proceso *form* usando un 70 a 80 % de la CPU.

Como trabajo futuro se propone continuar modularizando el código separándolo en las clases: PV, Diesel, Historizador, Baterías, EMS y Planificación . También se plantea indagar más en las estructuras de control para el control de la microrred y cómo sus componentes se coordinan. También se propone migrar al lenguaje Python por su popularidad y por ser de código abierto.

Agradecimientos

Quiero dedicarle este trabajo a mi madre Luz, mi padre Raúl y mi hermano Claudio, que con su ejemplo y trabajo imparable me inspiran a ser mejor cada día. A mi abuela Gladys y a mi abuelo Adrián por apoyarme siempre con sus palabras y actos y ser un eje fundamental en mi vida desde pequeño como segundos mapadres.

Hacer mención especial a mis amigos de la FAE, a Erick, Felipe y Nicolás por contenerme y apoyarme en estos años difíciles donde he conocido a la vida misma, quien soy y lo que quiero para mi futuro. A Roberto, amigo que conocí dando mis primeros pasos extra universitarios en las ERNC y con quien hemos compartido experiencias que nos han hecho crecer mucho los últimos años. Quiero dedicarle también este trabajo a mi amigo Gabriel, cuyo recuerdo y cariño atesoro mucho junto al de los amigos y amigas que conocí en esta etapa universitaria.

Quiero agradecer además al centro terapéutico Manto Wasi, en especial a Rumi, Ingrid y Juan, por acogerme y mostrarme el camino en momentos en que me encontraba totalmente perdido. A Juan en particular por convertirse en un cercano amigo y compañero de camino. A Nicolás V. por ayudarme a desenredar el complejo enmarañado que significó modularizar el programa que revisé en esta memoria. A Belén B. por mostrarme con el ejemplo como ser más ordenado, estar ahí en momentos difíciles y decirme las cosas que necesitaba escuchar, con su manera particular y especial. A mis amigas Josefina, Camila y Valentina, por estar emocionalmente en los momentos en que más lo necesité cuando volví a la universidad a terminar mi carrera. A Javiera, Angélica y Carolina, por su ejemplo de gestión emocional para enfrentar las relaciones y la vida. De alguna manera también agradecer a todas las personas que por algún motivo se cruzaron en mi camino y me enseñaron cosas que necesitaba aprender.

Quiero agradecer por supuesto al profesor Patricio Mendoza A. por su participación como guía de esta memoria y su paciencia conmigo incluso en los momentos más complicados; a Fernando Lanas, cuya ayuda alivió más de una pesadilla nocturna con la tesis y al profesor Andrés Caba, por sus aprietes constantes para que mi trabajo se ajustara a los estándares de la Universidad.

Finalmente agradecer a la Universidad de Chile por haberme becado en mis primeros años, beca que alivió mis bolsillos y los de mis ma-padres y me permitió estudiar, en un país donde en la actualidad las oportunidades dependen más de tu ingreso y donde naciste que de tus capacidades, y que como estudiante de primera generación agradezco de corazón.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Motivación

El aumento de los DER (Distributed Energy Resorces), los sistemas de almacenamiento y las cargas controlables están promoviendo la capacidad de construir microrredes a partir de la red de distribución actualmente disponible o bien en zonas aisladas donde esta red no llega.

En esta memoria se busca abordar los aspectos necesarios para proponer un rediseño del sistema de control SCADA para la microrred implementada en Huatacondo. Para ello se abordan tantos aspectos técnicos del control para entender las formas de funcionamiento, como aspectos referentes a la modularización del software mismo.

Como objetivo específico se abordarán el modularizar un archivo del actual sistema SCADA implementado en Huatacondo, que funciona en la plataforma Visual Estudio. Actualmente el sistema implementado tiene imbricadas todas sus componentes y al caerse un proceso, todo el sistema cae. Para mejorar este prblema, la propuesta es modularizar de forma que cada proceso sea independiente entre sí, conectándose entre ellos a través de funciones. De esta forma se dota de independencia a cada proceso para que el sistema opere de mejor forma. El archivo al cual se le aplicará esta modularización lleva por nombre *"form1"*, y en él se separarán las componentes del Backend de las del FrontEnd.

En la propuesta se postula trabajar con el mismo lenguaje en cual está programado el SCADA, C#, un lenguaje orientado a objetos como Java, C++ o Python, ya que es poco realista transitar a otro lenguaje dentro del período que se debe realizar el trabajo.

La motivación de este trabajo nace de la necesidad de contar con un programa SCADA que controle la microrred de Huatacondo que sea robusto, le de confiabilidad al sistema y tenga menos caídas en su funcionamiento, apuntando a operar de la forma más eficiente y económica posible. Los criterios abarcados anteriormente se resumen en lo que se llama "estabilidad del programa".

1.2. Objetivos

1.2.1. Objetivo General

El trabajo de memoria "Modularización de software SCADA para la microrred implementada en Huatacondo usando lenguaje C#", aborda principalmente la modularización del código del software SCADA implementado en la actualidad en la microrred de Huatacondo.

El objetivo general es realizar una propuesta de plan de modularización del código del software SCADA actual, en el cual operan actualmente los módulos que controlan la demanda, la generación y los datos de manera conjunta, de manera tal de separar el programa en partes funcionalmente independientes que encapsulen operaciones y datos, con miras a mejorar el desempeño del programa.

1.2.2. Objetivos Específicos

1. Revisar el estado del arte de las arquitecturas de software más utilizadas a nivel mundial para abordar el problema de control de un SCADA
2. Mejorar el desempeño del sistema mediante la modularización del SCADA actual implementado en Huatacondo.
3. Proponer un esquema de modularización, separando estructuralmente el archivo *"form1"* en FrontEnd y Backend
4. Probar si el desempeño mejora o empeora con herramientas de medición del programa Visual Studio.

1.2.3. Alcances de la memoria

1. Se trabajará con datos extraídos del funcionamiento de la microrred de Huatacondo
2. Se utilizarán diagramas de clase extraídos del programa mismo para estudiar su modularización
3. Solo se trabajará con el lenguaje C#, proponiendo para un trabajo futuro la migración del software a uno programado en Python.
4. Se trabajará solo con el software del SCADA asociado a la microrred.
5. Se dejará algunas partes del programa, en particular el EMS, fuera de funcionamiento para las pruebas del programa.
6. Se analizará el diagrama de clases del software SCADA desarrollado en función de diagnosticar el problema
7. Se propondrá un esquema de trabajo con pasos para modularizar el programa.
8. Se propondrá una prueba para estudiar el desempeño del SCADA
9. Se trabajará en separar el archivo *"form1"* en un FrontEnd y un Backend.
10. Se probará y analizará el funcionamiento de 4 casos: caso base sin modularizar, interactuando y sin interactuar; y caso modularizado, interactuando y sin interactuar.

No se encuentra dentro de los alcances de esta memoria la conexión con la red real, por lo que la medición de datos y el envió de consignas de la red no se desarrollara en este trabajo.

El desarrollo en el sistema SCADA se realizará solo hasta el paso de la separación entre Backend y FrontEnd, lo demás quedará como trabajo futuro. Se documentará debidamente el proceso.

Tampoco se encuentra dentro de los alcances de esta memoria la conexión del programa a la red de Huatacondo. Para probar el SCADA, se realizará una simulación de datos de entrada para el software, de manera de poder probarlo con la herramienta *"Performance Wizard"* disponible en el menú *"Analyze"* de Visual Studio. Para todos los casos se ha desactivado el EMS, de manera que el programa no lo corre y se probará sin esa funcionalidad.

1.2.4. Estructura del Trabajo

A continuación, se presenta brevemente el contenido de cada un de los capítulos desarrollados en el trabajo de título.

En el capítulo 2 se presenta el marco teórico del trabajo en el cual se revisa toda la información disponible respecto a las microrredes y los sistemas SCADA ; se describen algunas estrategias de control y tipos de arquitecturas de control. Además se hace una revisión de los lenguajes de programación orientados a objetos, modularización de programas y diagramas UML.

En el capítulo 3, se detalla y explica el tipo de modularización que se utilizará para el sistema en cuestión y como se quiere aplicar este al código. Además se detallarán las pruebas que se harán y los casos abordados.

En el capítulo 4 se presentan los resultados, el análisis de esto y la discusión al respecto

Finalmente en el capítulo 5 se presentan las conclusiones.

En el anexo se incluye código relevante, el diagrama UML en tamaño más grande y además algunos errores de ejecución que podrían ser de interés para el lector.

Capítulo 2

Marco Teórico

2.1. Sistema SCADA

Un sistema SCADA(acrónimo de Supervisory Control And Data Adquisition), es un software de Supervisión, Control y Adquisición de Datos, el cual permite controlar y supervisar a la vez, procesos a distancia.

Esto permite la retroalimentación en tiempo real con los instrumentos de campo (sensores y actuadores), y controla el proceso automáticamente. Además, el sistema almacena la información que recoge durante todo el proceso, en la base de datos, y permite a su vez que esta se gestione e intervenga.

De acuerdo a la teoría de control, retroalimentación es el proceso por el cual, la señal de salida vuelve a incidir en la entrada. De esta manera se mantiene el control sobre el comportamiento dinámico del sistema.

En particular, en esta memoria se abordará el estudio de sistemas SCADA enfocados en el control y operación de microrredes aisladas. Esta tecnología, los sistemas SCADA, si bien no es nueva, las arquitecturas de funcionamiento y la operación de las microrredes aún no poseen un estándar.

2.1.1. Sistemas de Lazo Cerrado

Los sistemas de lazo cerrado, a diferencia de los sistemas de control de lazo abierto, tienen como característica principal que existe una retroalimentación desde la señal de salida del sistema hacia la entrada. Esto se utiliza para comparar estas dos señales e ir ajustando la señal de salida en función del error entre la referencia y la señal generada. En general, los sistemas orientados a controlar dentro de un rango un parámetro como por ejemplo temperatura, velocidad, presión, caudal, fuerza, posición, entre otras, son de lazo cerrado debido a la idea de controlar hasta que .[el] error"se haga cero o menor a un valor.

Realimentación

La realimentación es el proceso por el cual el sistema captura información del proceso, para que esta pueda ser analizada en función de obtener indicadores o valores que permitan mejorar el proceso. Estos pueden ser:

- Indicadores sin realimentación inherente (no afecta el aproceso, solamente al operador)
  - Estado actual. Valores instantáneos. Ej: rapidez de un vehículo; caudal de riego; etc.
  - Desviación o derivada del proceso. Evolución histórica y acumulada.
    - Medición de parámetros
- Indicadores con retroalimentación inherente (afectan al proceso y luego al operador):
  - Generación de Alarmas
  - HMI Human Machine Interface (Interface hombre-máquina)
  - Toma de decisiones
    - Mediante operatoria humana
    - Automática, mediante algoritmos que programen acciones.

2.1.2. Esquema general de un SCADA

Un sistema SCADA está conformado por un hardware con señal de entrada y salida, controladores, una interfaz humano-máquina (HMI), redes, comunicaciones, base de datos y software. A continuación se muestran unas imágenes de ejemplo del SCADA implementado en el laboratorio DECC [8].

Los componentes del sistema son principalmente 3:

- Varias Unidades de Terminal Remota u Estaciones Externas (RTU ó UTR)
- Estación Maestra y Computador con HMI
- Infraestructura de Comunicación

Cada una soporta a las otras y funcionan interconectadas. El SCADA tiene una parte de software y otra de hardware.

2.2. Energy Management System

Un EMS(del inglés Energy Management System), es un sistema de herramientas asistidas por computadora que utilizan los operadores de las redes eléctricas para monitorear, controlar y optimizar el rendimiento del sistema de generación y / o transmisión, en otras palabras, se trata de un Sistema de Gestión de Energía. Además, este se utiliza en la actualidad en sistemas de pequeña escala como microrredes [9] [10]

La tecnología informática también se conoce como SCADA / EMS o EMS / SCADA. En este sentido, la terminología EMS excluye las funciones de monitoreo y control, pero más específicamente se refiere al conjunto colectivo de aplicaciones de red eléctrica y a las aplicaciones de control y programación de generación.

Los fabricantes de EMS también suelen suministrar un simulador de entrenamiento de despachador correspondiente (DTS). Esta tecnología relacionada hace uso de componentes de SCADA y EMS como una herramienta de capacitación para operadores de centros de control.

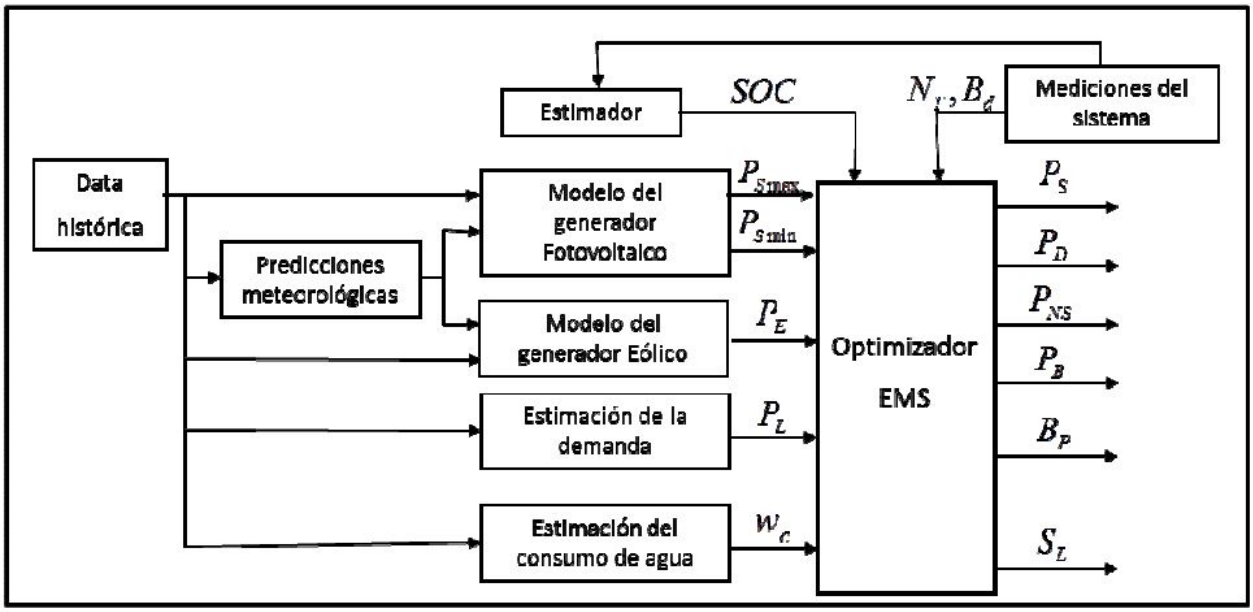
Los sistemas comerciales de gestión de la energía también suelen ser utilizados por entidades comerciales individuales para monitorear, medir y controlar las cargas eléctricas de sus edificios. Los sistemas de administración de energía se pueden usar para controlar centralmente dispositivos como unidades HVAC y sistemas de iluminación en múltiples ubicaciones, como tiendas, supermercados y restaurantes. Los sistemas de administración de energía también pueden proporcionar funciones de medición, submedición y monitoreo que permiten a los administradores de instalaciones y edificios recopilar datos y conocimientos que les permitan tomar decisiones más informadas sobre las actividades de energía en sus sitios. A su vez estas decisiones pueden funcionar bajo un esquema automático de optimización, en el cual los datos de entrada (dispositivos, mediciones, etc), se utilicen para optimizar lo que necesite el sistema. Ej: optimizar la cantidad de energía utilizada en una microrred.

En este contexto, el año 2011, F. Lanas propone en su memoria [1] un Sistema de Gestión de Energía de Control Centralizado para la operación automatizada de la microrred de Huatacondo, la cual utiliza un horizonte deslizante realizando un despacho de las unidades disponibles cada 15 minutos, con el objeto de ajustarse mejor a las condiciones de viento, sol, demanda y energía en la batería.

Para el correcto funcionamiento de este EMS en la microrred se requiere predicción de los consumos residenciales (tanto eléctricos como de agua), predicciones de la potencia fotovoltaica que generara la planta fotovoltaica y predicciones de la potencia eólica que generara el aerogenerador para las próximas 48 horas, esto dado que se realizará una optimización y su consecuente predespacho para las próximas 48 horas. Además se requiere conocer el estado de carga del sistema de baterías y el estado prendido/apagado del generador diesel al momento de realizar la optimización. Con toda esta información, se podrá realizar una optimización del despacho de las unidades generadoras para minimizar los costos operacionales de la microrred.

En la siguiente figura acontinuación se presente el anterior modelo propuesto:

Figura 2.1: Esquema simplificado del EMS propuesto en la memoria de F. Lanas [1]



2.3. Microrred

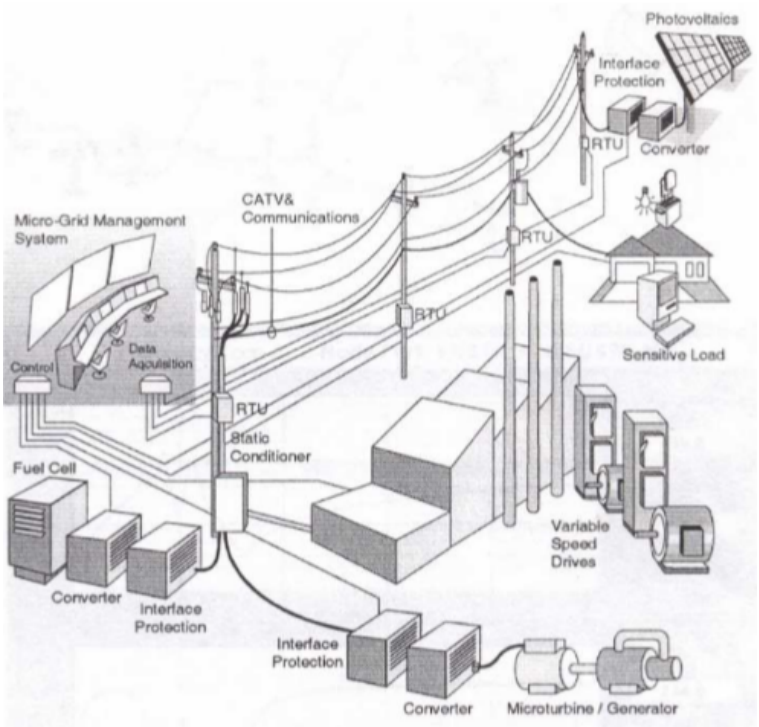
Una microrred es una pequeña red electrica que funciona con pequeños medios distribuidos de generación de energía. El concepto de microrred engloba la idea de que el consumo de energía y la producción de esta estén cercanos fisicamente, de manera de reducir las pérdidas en transmisión de potencia, al mismo tiempo que las fuentes de generación locales impactan en menor manera sus alrededores; a esta forma de generación eléctrica con pequeñas fuentes locales se le llama "Generación Distribuida".[1] Las microrredes pueden definirse como una red con baja tensión, por ejemplo en un pueblo o una área urbanizada, junto con sus consumos y múltiples unidades pequeñas de generación conectadas a ella, con potencias que van en los rangos de algunas decenas de kW o menos. Estas pueden funcionar ya sea de forma aislada o bien conectada con otra red. El modo isla (aislado) sucede cuando la red se separa de la red principal y se alimenta solo desde las unidades de generación distribuidas ubicadas en ella misma.

Típicamente una microrred incluye los siguientes componentes:

- Múltiples alimentadores suministrando energía a los consumos
- Sistemas de pequeña y micro-generación
- Sistemas de almacenamiento de energía eléctrica
- Un sistemas de control y gestión apoyado por una infraestructura de comunicaciones

A continuación se muestra un esquema de una microrred tipica en la Figura 2.2.

Figura 2.2: Micro red típica



Uno de los aspectos importantes en una microrred y en cualquier sistema eléctrico de potencia es la confiabilidad de este, la cual mejora gracias a la Generación Distribuida (GD) ya que en caso de caídas de máquinas o corte eléctrico, las unidades de generación pueden seguir alimentando consumos locales que en algunos casos podrían ser de extrema necesidad, como por ejemplo hospitales o lugares donde se realice la conservación de alimentos. Además, existe la posibilidad de que la microrred opere de manera aislada[11], para lo cual se requiere contar con elementos de protecciones, componentes de control y comunicación que permitan una operación segura del sistema; luego, el principal desafío de operar este tipo de sistema es la coordinación de las distintas unidades de generación, consumos y almacenamientos, y poder cumplir los requerimientos técnicos de la microrred como la potencia activa y reactiva[12], voltaje y frecuencia[12].

Existen diferentes métodos de control: por frecuencia de la red[13], a través de agentes (controlando los componentes del sistema), a través del control primario, secundario y terciario[14]; también a partir de 4 etapas de funcionamiento: Forecasting, Planning, Online replanning and Local commmand, etc. Si bien estos métodos permiten el control de la red, no aseguran una operación económica [15] [16] y segura del sistema. Para ello, se deben considerar nuevos métodos y estructuras que permitan una operación y control inteligente del sistema distribuido constituido por diferentes unidades de generación [11], consumos y almacenamiento[17].

Tabla 2.1: Modo de conexión de microrredes

Aislado y Conectado a la Red - Común	**Sólo modo isla**	**Sólo conectado a la red**
Conexión/Desconexión Suave de la Red	Control de Frecuencia	Corrección del Factor de Potencia
Programación de Generadores	Control de Voltaje	"Peak Shaving"
Generador/Almacenamiento kW compartiendo el uso de Energía Renovable	Soporte de Sobrecarga del Generador	Zero Grid feed-in power height

2.3.1. Experiencia de Microrred en Huatacondo

Huatacondo se encuentra cercano a un oasis ubicado a 230 kilómetros al sureste de Iquique, en la Región de Tarapacá. Está ubicado e la vera del antiguo Camino del Inca de Arica de Quillagua, y en medide la Pamapa del Tamarugal. Adempás se encuentra aislado de la red eléctrica, siendo la línea que más cerca pasa está a más de 145 kilómetros de distancia del poblado.

A continuación, en la figura siguietnte se observa una imagen satelital de la posición geográfica del poblado de Huatacondo [1]

Figura 2.3: Imagen Satelital vista continental de la ubicación de Huatacondo

Figura 2.4: Imagen Satelital vista cercana de la ubicación de Huatacondo. Fuente: Google Earth Pro.

Esta localidad posee las siguientes características específicas:

- la población no sobrepasa los 100 habitantes
- El sistema se encuentra aislado del Sistema Interconectado Nacional (SIN, ex fragmento del antiguo SING), por lo que debe autoabastecerse de energía.
- El pueblo tenía solo 10 horas al día de electricidad, abastecido por un generador diesel. Este funciona de fora limitada por el alto costo del combustible que el generador consume.
- Los recursos renovsables básicos disponibles son viento y radiación solar. Se observa la presencia aprovechable de biomasa y energía hidráulica en pequeñas cantidades, para un posile desarrollo futuro.

Tabla 2.2: Parámetros de las Unidades operando en Huatacondo. Fuente: [1]

Parámetros	**Valor**	**Unidad**
Potencia Panel Fotovoltaico	22	kW
Potencia Turbina Eólica	2,5	kW
Potencia Máxima del acumulador	40	kW
Capacidad del Acumulador	150	kWh
Potencia Máxima del Grupo Electrógeno	120	kW
Potencia Mínima del Grupo Electróngeno	10	kW
Bomba de agua	1-2	HP
Volumen Máximo del estanque de agua	16000	l
Volumen Mínimo del estanque de agua	1600	l

Para solucionar el problema de abastecimiento de Huatacondo, se planificó con anterioriedad que se instalaran paneles fotovoltaicos con una potencia de 22 [kW] y una turbina eólica con una potencia total de 2.5 [kW], la cual no está actualmente operando en la microrred. Para almacenar la energía eólica de las fuentes de ERNC, se instalará un acumulador de 150 [kWh] de capacidad y potencia máxima de 40 [kW].

El pueblo cuenta ya con un grupo electrógeno, con potencia mínimma de 10 [kW] y máxima de 120 [kW], con el cual se abastece actualmente. Además, el pueblo hace uso de una bomba de agua con la cul alimentan un estanque de agua para uso local. Este estanque se determinó tendría un volumen máximo de 16.000 [l] y mínimo de 1.600 [l], por lo cual la bomba también deberá ser controlada.

La información anterior se encuentra resumida en la siguiente tabla :

El problema que presentan las fuentes de ERNC es su poca confiabilidad al satisfacer la entrega de potencia, ya que las fuentes de energía dependen de condiciones climáticas que varian en cada minuto. Debido a estas fluctuaciones, no es posible alimentar un poblado sin ejercer algun tipo de coordinación sobre los distintos generadores eléctricos.

Sistema de Gestión de Energía (EMS) implementado en Huatacondo

En el software SCADA implementado en Huatacondo se propone una solución a través de un Sistema de Gestión de Energía (en inglés Energy Management System ó EMS) que optimice la operación de un conjunto de unidades de generación y cargas, uilizando sistemas de comunicación, monitoreo y control para aplicar el predespacho.

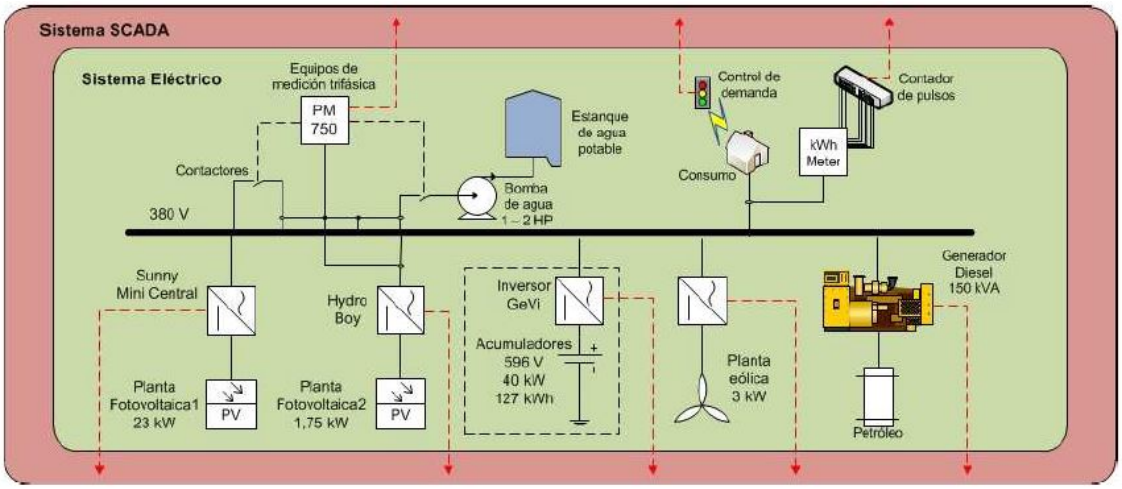
Este EMS deberá considerar y evaluar el funcionamiento de los siguientes componentes:

1. Bomba de agua
2. Estanque de agua
3. Control de la Demanda
4. Consumo residencial

5. Generador Diesel
6. Generación de la Planta Eólica
7. Generación de planta fotovoltaica
8. Banco de Baterías

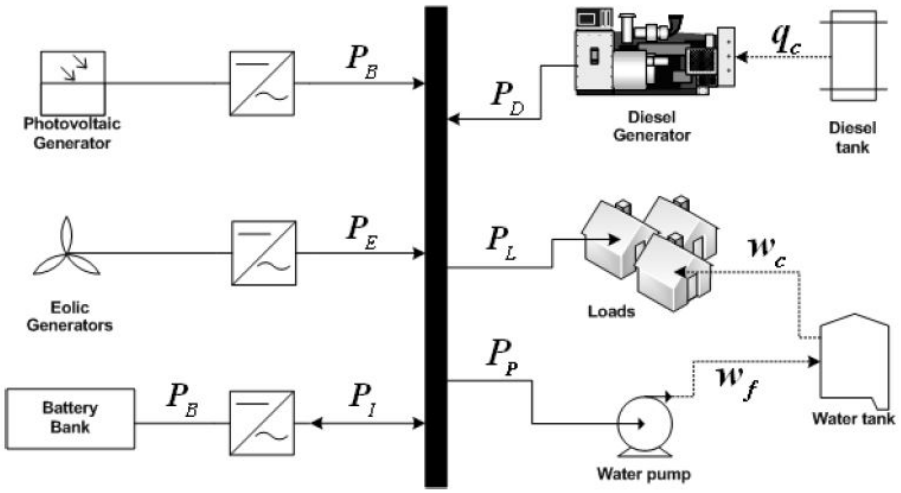
Esto será monitoreado por un sistema SCADA, El modelo de monitoreo de los equipos y cargas que se presentan en Huatacondo se indica en la Figura 2.5, la cual corresponde a un diagrama Unilineal de Fuerza de la red del poblado.

Figura 2.5: Diagrama Unilineal de Fuerza de Huatacondo, fuente: Centro de Energía, FCFM



La siguiente figura resume la conexión de las unidades que componen esta microrred en la cual se tiene: un generador fotovoltaico, un aerogenerador, un banco de baterías, un generador diesel, una bomba de agua y los consumos residenciales. En la actualidad si bien el generador eólico no se encuentra operando, se añadió de igual forma en la Figura 2.6 de manera representativa.

Figura 2.6: Microrred de Huatacondo, fuente [12]



2.3.2. Unidades de Generación y Almacenamiento

Generación solar

El sol es una de las principales fuentes de energía que alimentan al planeta Tierra y que permite que el ciclo de la vida siga ocurriendo desde los principios de la humanidad . Es la fuente y el origen de todas las otras fuentes de energías como la eólica, hidráulica y la biomasa.

La Tierra recibe petawatts (10$\hat{1}$5 watts) de radiación solar entrante desde la capa más alta de la atmósfera. Aproximadamente el 30 % es reflejada de vuelta al espacio mientras que el resto es absorbida por las nubes, lo océanos y las masas terrestres.

La potencia de la radiación varía según el momento del día, las condiciones atmosféricas que la amortiguan y la latitud. A esta potencia sobre la superficie terrestre se la conoce como irradiancia, definida como el valor de la intensidad energética promedio de una onda eletromagnética en un punto dado y se calcula como elvalor promedio del vector dde Poynting.

La irradiancia sirve de base para la definición de radiancia, la energía emitida por unidad de superficie y por unidad de ángulo sólido. Rambién, se le utiliza para definir la constante solar, la que corresponde a la irrancias sobre un plano ubicado en la superficie de la atmósfera, sobre el cual los rayos solars inciden normalmente. Su valor es de 1367 W/m^2 segunla escala del World Radiation Reference Centre (WRRC), de 1373 W/m^2 segun la Organización Mundial de Metereología (WMO de sus siglas en inglés) o de 1353 W/m^2 segun la NASA.

En la actualidad, es posible utilizar paneles fotovoltaicos para captar esta radiación y convertirla en energía eléctrica. Un panel fotovoltaico está compuesto por numerosas celdas que convierten la luz en electricidad. Estas celdas funcionan en base al efecto fotovoltaico, por el que la energía luminosa produce cargas positiva y negativas en dos seminconductores próximos de diferente tipo, produciendo así un campo eléctrico capaz de generar una corriente.

Generación Eólica

La energía eólica es considerada como una forma indirecta de energía solar, producida por el movimiento del aire, ocasionando por la diferencia de temperaturas en la superficie terrestre, que origina el desplazamiento de masas de aire que paseen energía cinética, la que es transformada en otras formas útiles para las actividades humanas. En la actualidad, la energía eólica es utilizada principalmente para producir energía eléctrica mediante aerogeneradores.

La energía eólica es un recurso renovable y limpio, sin embargo, el principal incoveniente es su aleatoriedad y variabilidad, las cuales no sólo dependen de las condiciones atmosféricas y climáticas, sino que también de las características geográficas del sector y la altura sobre e nivel del suelo.

Los movimientos de las masas de aire tienden a compensar las diferencias de presión exisentes, de modo que, a mayor gradiente d presión es mayor la veloidad del viento. El

movimineto de las mesas de aire va desde las altas presiones a las bajas presiones.

Otro punto ue se debe tener en cuenta es la variación de la velocidad horaria del viento; durante el día el vient sopla más fuerte que en la noche. Esto es producido por el calentamiento de las masas de aire en las horas con sol, lo que genera una atmósfera inestable. Durante la noche se presenta una atmósfera estable, ya que los gradientes de temperatura no son considerables.

La energía cinética de una masa de aire **m** moviéndose a una velocidad **v** se encuentra dada por la siguiente expresión:

$$E = \frac{1}{2}mv^2 \tag{2.1}$$

La energía cinética por unidad de volumen será:

$$\mathrm{e} = \frac{1}{2}pv^2 \tag{2.2}$$

Donde p es la densidad del aire (se puede suponer constante). Luego, si el flujo de aire que atraviesa una superficie A es:

$$\phi = v \cdot A \tag{2.3}$$

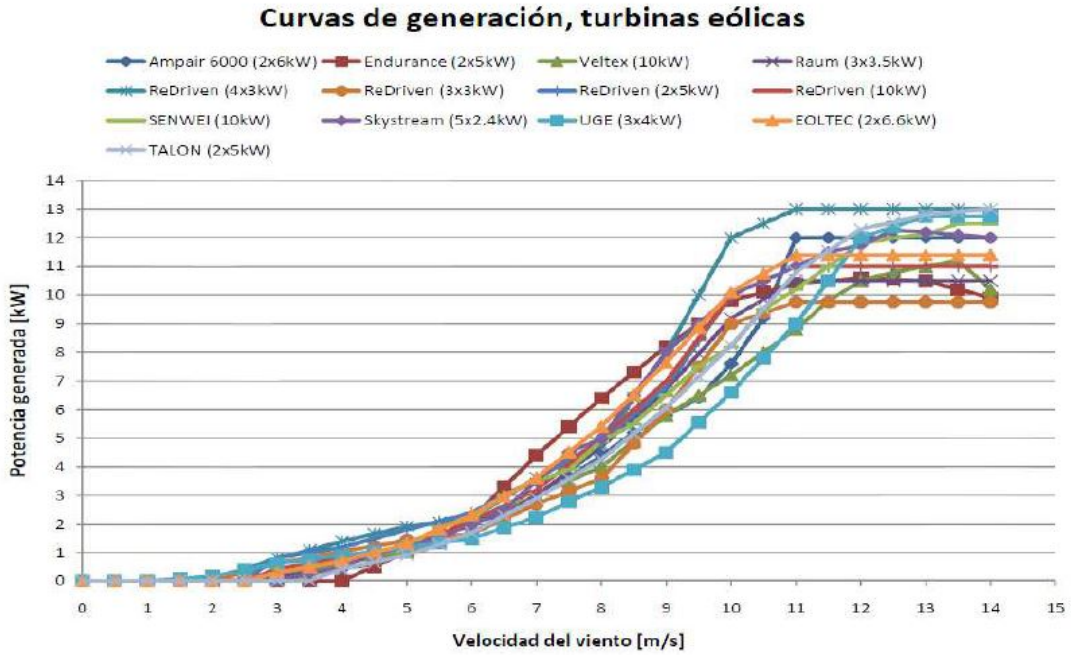
Si la masa de aire atraviesa un área **A** (área del rotor de un aerogenerador) perpendicular a la corriente del aire, con una velocidad **v**, posee una potencia cinética:

$$P_{viento} = \frac{1}{2}p \cdot v^3 A[W] \tag{2.4}$$

Se debe recalcar que la potencia del viento es proporcional al cubo de la velocidad [18]

Para aprovechar esta energía cinética, se utiliza la tecnología de los aerogeneradores, los que básicamente mediante el giro de la turbina hacen girar el rotor de un generador produciendo electricidad. los fabricante de aerogeneradores proporcionan la curva de aerogenerador donde se indica la potenia elécgtrica disponible a diferentes velocidades del viento. Ésta es utilizada para calcular la potencia extraíble; cada aerogenerador tiene su propia curva, la que depende de sus características constructivas. En la siguiente figura 2.7 se pueden observar las curvas de distintas turbinas eólicas que fueron considerados para este proyecto. La que se utilizó finalmente fue una turbina eólica SENWEI de 2.5 [kW].

Figura 2.7: Curvas de Generación para distintos Aerogeneradores, fuente: [12]*



Modelos fenomenológicos

La modelación numérica de la atmósfera utilizando modelos globales y regionales (Numerical Weather Prediction o NWP) ha alcanzado en los últimos años un nivel aceptable para zonas con terreno llano o moderadamente complejo. Sin embargo, los resultados siguen siendo imprecisas en zonas de orografía compleja o cuando los fenómenos atmosféricos que intervienen son complicados.

Con el fin de solucionar esta dificultad y poder disponer de predicciones con una resolución espacial susceptibles de ser utilizadas para pronosticar la energía eólica se puede recurrira un modelamiento de mesoescala. Esta técnica consiste en la utilización de uno o arios modelos físicos encadenados que, alimentándose de los resultados de un modelo numérico de predicción metereológica global o de ára limitada, se ejecutan para dar predicciones con mayor resolución espacial. Esta resolución puede ser de algunso centenares de metros, en comparación a las decenas de kilómetros de resolución de NWP. Dado que el dominio de aplicación de estos modelos es mucho menor que el del modelo global, son capaces de simular un mayor número de procesos físicos, con lo cual pueden considerar un terreno más cercano a la realidad, dando lugar a pronósticos más preciosos, especialmente en lo que respecta al viento.

Los modelos físicos tienen un pobre desempeño en plazos muy cortos de predicción, esto es, desde segundos hasta aproximadamente 6 horas hasta incluso una semana; de esta manera se logra un desempeño bastante aceptable hast las 42 horas aproximadamente[19].

Modelos estadísticos

Los modelos estadísticos no simulan los procesos físicos explícitamente, sino que busca una relación entre los pronósticos meteorológicos y las producciones de potencia eólica.

La principal ventaja de este tipo de modelos es que requieren un menor coste computacional al no tener quje simular matemáticamnte los complejos fenómenos físicos que influye n en la predicción de la producción de energía eólica y sus interrelaciones. Sin embargo, los modelos estadísticos requieren de una amplia base de datos históricos para su entrenamiento.

La Figura 2.8 indica en color gris oscuro el rango de NMAE (Normalized Mean Absolute Error) esperado para distintos predictores típicos utilizados por empresas que prestan servicios de predicción de potencia de un parque eólico. El NMAE se define como:

$$NMAE = \frac{\sum_{n}^{t=1}(|\hat{\theta}(t) - \theta(t)|)}{n \cdot \bar{\theta}(t)} \tag{2.5}$$

Donde $\hat{\theta}$ es el valor predicho, θ el valor observado, n es el número de observaciones y $\bar{\theta}$ es el valor medio observado.

Estos predictores normalmente utilizan una mezcla de modelos físicos y estadísticos, rescatando las mejores predicciones de cada uno de ellos. Se puede obsr var un aumetno estrictamente cerciente de los errores de predicción en función del tiempo.

2.3.3. Banco de Baterías

Baterías, o acumuladores, es el nombre con el que se conoce a los dispositivos que almacenan energía eléctrica usando procedimientos electroquímicos y que pueden devolverla posteriormente. Este ciclo puede repetirse por un determinado número de veces luego del cual, generalmente al fin del ciclo de su vida útil, deberá cambiarse la batería por una nueva. El número de ciclos depende de la tecnología en la que se basa el acumulador. Para el caso de baterías de plomo-ácido de descarga profunda, como las que se utilizaran en Huatacondo, este valor varía entre 1.000 a 10.000 usos.

Circuito equivalente y ecuaciones características

El circuito equivalente de un acumulador puede ser más o menos complicado según las condiciones de operación y tipo d acumulador. Los parámetros característicos más significativos en régimen permanente de corriente continua, o de bajas frecuencias, son la tensión electroquímica natural U_{e}, la resistencia interna del electrolito al paso de los iones durante los procesos de descarga R_{d}, la resistencia interna del electrolito durante el proceso de recarga R_r. En menor grado, la resistencia eléctrica equivalente representativa de los fenómenos de electrolisis no deseados R_{H2O} y, también, la resistencia eléctrica representativa de las fugas

Figura 2.8: Rango típicos de predicciones para un parque eólico, fuente: [2]

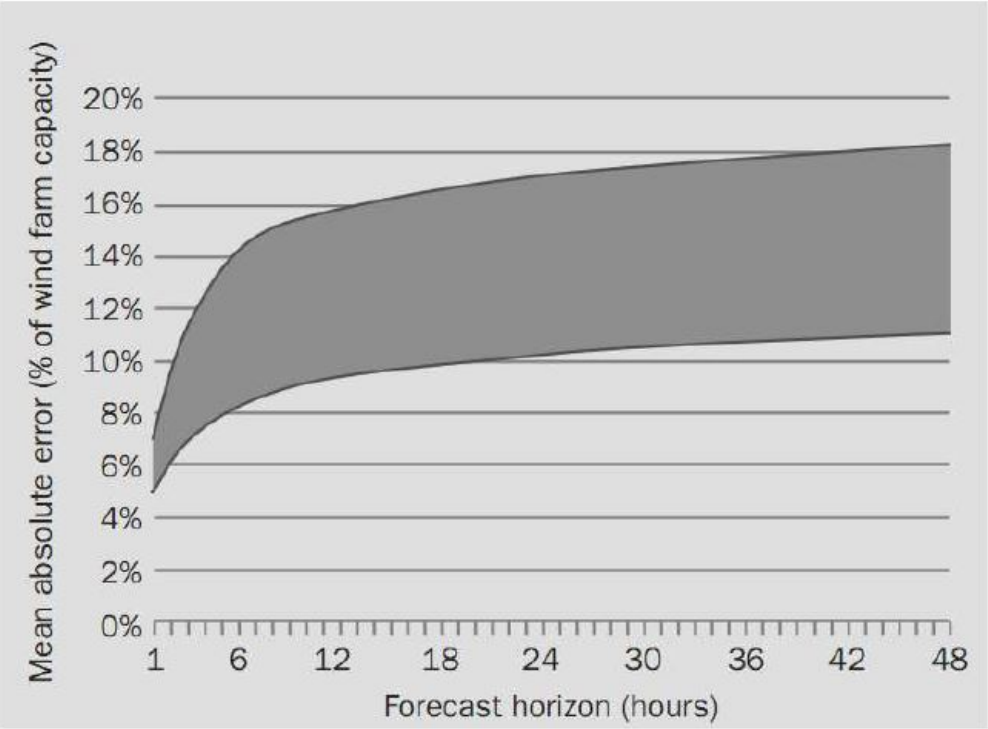


Figura 11 : Rangos típicos de precisión de predicciones para una parque eólico, fuente: [17]

naturales a través del solvente, carcasa y estructuras de sujeción R_{fug}. El circuito equivalente puede verse en

Resistencia del Electrolito a la descarga R_{d} y a la recarga R_r

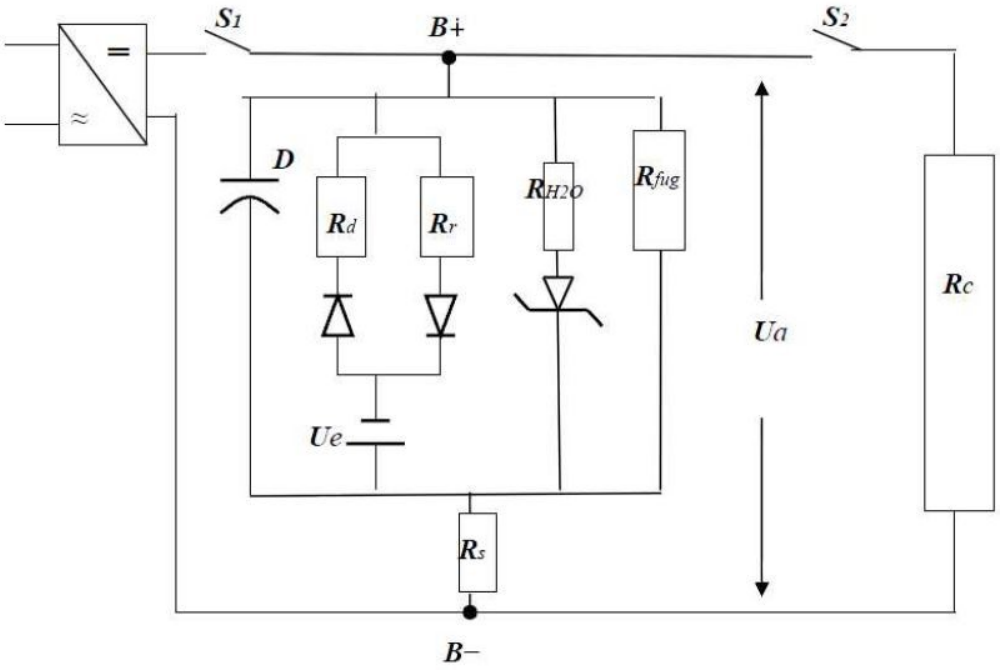
Durante los procesos de descarga y recarga los iones del sistema electroquímico se aceleran por efecto del campo, migrando al interior del electrolito, colisionando a su paso con las moléculas del solvente. En este proceso transfieren energía cinética al medio, calentándolo: Se consume energía.

Desde el punto de vista eléctrico, estas pérdidas se representan mediante una resistencia: la resistencia interna del electrolito. Su valor depende de la conductividad l característica del electrolito y de la geometría de los polos del acumulador.

Para el caso de acumuladores compuestos por placas paralelas, la resistencia R del electrolito, en primera aproximación (despreciando los efectos de borde), queda determinada por la superficie conductora S efectiva útil de las placas y la separación o distancia d entre las placas de distinta polaridad, según la siguiente expresión:

$$R = \frac{\mathrm{d}}{\lambda \cdot S} \tag{2.6}$$

Figura 2.9: Circuito equivalente de una Batería de Plomo-ácido. Fuente: [3]



2.4. Arquitecturas de Red y Control

Una arquitectura de red es el diseño constructivo para organizar distintos componentes que conforman una red de nodos.

En cuanto a cómo estas estructuras afectan el control de los sistemas, existen muchas formas de organizar los sistemas. Hay sistemas que funcionan de forma centralizada, concentrando todos sus recursos en un nodo, ubicado en un nivel superior en materia de decisiones, que permite coordinar el sistema. También existen sistemas descentralizados cuyo control recae sobre todos los nodos del sistema (aquí tenemos ejemplos como la tecnología Block-Chain) y finalmente, existen híbridos entre ambas formas de controlEn el caso del control centralizado, existe una entidad que opera y coordina todos los elementos, mientras que en el control descentralizado cada uno atiende y responde a las señales del sistema de forma independiente pero coordinada; una mezcla de ambos delega algunas decisiones a cada nodo del sistema (ya sea consumo, carga o almacenamiento), mientras que otras decisiones como la operación económica vienen dadas por un coordinador centralizado.

2.4.1. Estrategias de Control

Sistema de control

Una microrred a través de su sistema de control, se encarga de asegurar el conjunto de funciones que mantienen a la red operando a mínimo costo, de manera segura y eficiente. Algunas de las funciones que también debe ayudar a controlar el sistema son: suplir la demanda, mantener niveles de servicio para cargas críticas, participar del mercado eléctrico, etc. Para lograr esto, el sistema puede estar orientado a un control centralizado como uno descentralizado [20]

La gestión convencional del sistema de potencia se suele realizar de forma centralizada. Esta es muy útil cuando se tiene un solo objetivo en especifico , en este caso la principal resposabildad recae sobre el Controlador Central de la Microrred (Microgrid Central Controller en inglés) [21]. La aproximación al problema desde el punto de vista centralizado es cada vez menos atractiva por sus sabidos inconvenientes: un úinico punto de falla y la necesidad de una alta capacidad de cómputo. En ambos controles se consideran 3 niveles jerárquicos:

1. Sistema de gestión de distribución (en inglés Distribution Management System o DMS)
2. Controlador central de microrred (en inglés MicroGrid Central Controller o MGCC) [22] o bien el Controlador Descentralizado de Microrred (en inglés Microgrid Descentralized Controller), según corresponda
3. Controladores Locales de microgeneradores (en inglés Microsource Controllers o MC) y controladores de cargas (en inglés Load Controllers o LC)

El DMS es un sistema de control de la red de distribución que coordina la operación con la red principal y el cual realiza principalmente las funciones de un SCADA: supervisión, control y adquisición de datos (Supervisory Control and Data Acquisition), pero además incorpora funciones que analizan el sistema de distribución y operaciones de soporte para las condiciones actuales y futuras. Cuando se trata de una microrred que opera de manera aislada solamente, este sistema no se hace necesario (más si el SCADA).

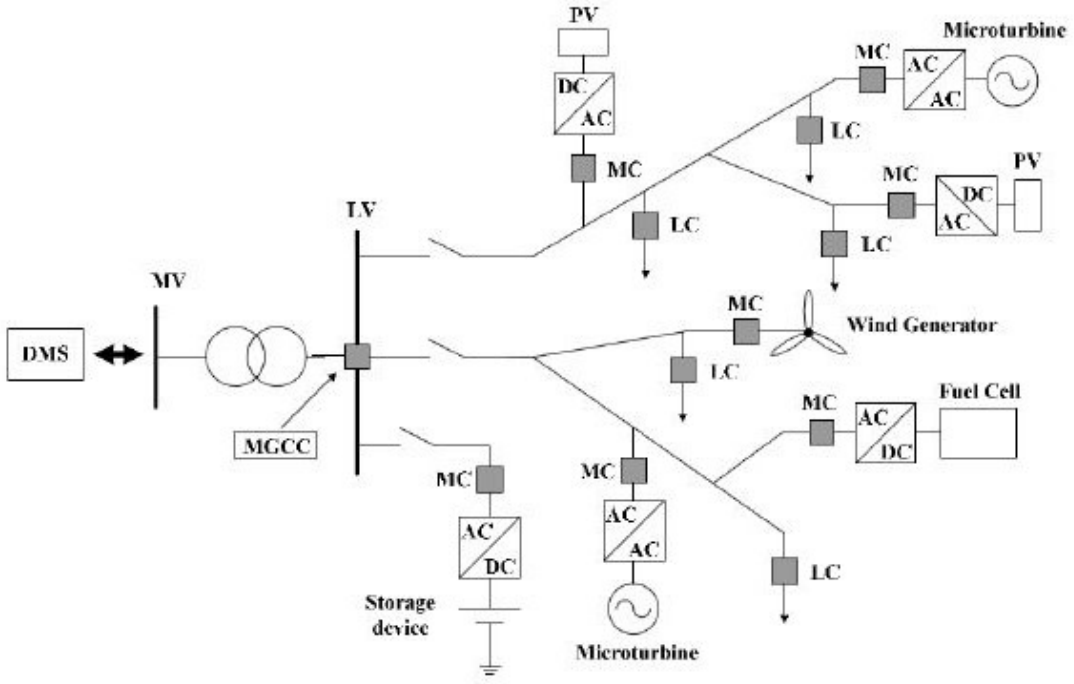
2.4.2. Control Jerárquico

El control jerárquico es un esquema de control donde estructuras inferiores en nivel de organización o producción están supeditadas a las decisiones de estructuras superiores. Por ejemplo, en un control jerárquico de una microrred implica que los procesos de los Controladores Locales están supeditados a un nivel de control superior donde sí se toman decisiones.

Estructura del sistema de control jerárquico

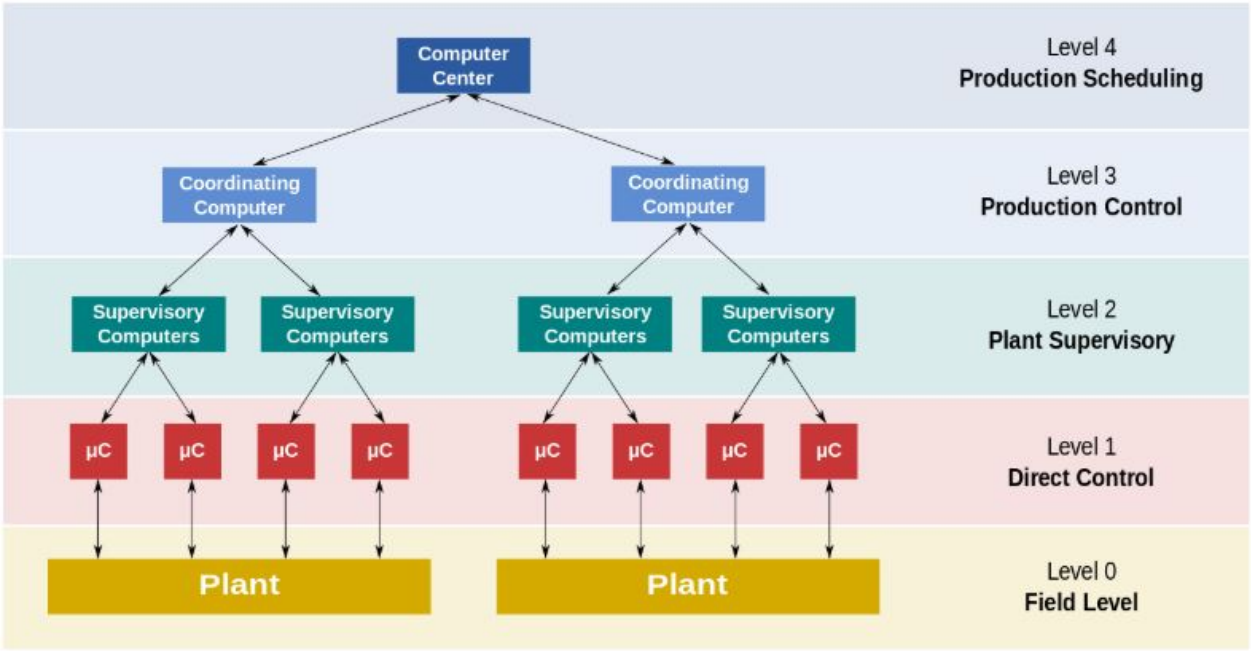
El sistema de control jerárquico tiene diferentes niveles. Cada uno representa una capa donde opera el sistema, en algunas se toman decisiones y en otras se opera simplemente. Las decisiones se toman a nivel de capas superiores en este tipo de control.

Figura 2.10: Arquitectura de una Microrred con Microgrid Central Controller (Controlador Centralizado de Microrred) Fuente: [4]



- El nivel 0 contiene los dispositivos de campos como los sensores que miden flujo y temperatura, y elementos finales de control como las válvulas de control.
- El nivel 1 continene los modulos industrializados Entrada/Salida (I/O), y sus procesadores electrónicos distribuidos asociados.
- El nivel 2 contiene los computadores supervisores los cuales recolectan la informacion, desde los nodos. procesadores en el sistema, y provee al operador pantallas de control.
- El nivel 3 es el nivel de control de producción,el cualno controla el proceso de forma directa pero le concierne el monitoreo de la producción y los objetivos.
- El nivel 4 es el nivel de planificacion de la producción.

Figura 2.11: Estructura General de un Sistema de Control Fuente: []



2.4.3. Control Descentralizado

Arquitectura Descentralizada

La arquitectura de control descentralizada para una microrred es propuesta, mejorando la confiabilidad del sistema y permitiendo la trasmisión de los comandos de control a través de la red[23].

Sin embargo, el control de microrrdes generalmente es más difícil que los tradicionales sistemas de potencia dado a la limitada capacidad de almacenamiento y la falta de inercia, una dinámica más rápida y el corto tiempo de respuesta de recursos distribuidos basados en inversores, y una alto grado de incertezas paramétricas y topológicas.

Una arquitectura realmente descentralizada de control debiese tener las siguientes propiedades[5]:

- **Descentralizada:** Dado este concepto, la arqueitectura descentralizada sugiere tener múltiples controladores locales de manera de lograr suavizar los transientes durante la operación y actuar como si el sistema tuviese un controlador central.
- **Compartir recursos:** cada controlador comparte el estado de su propio DER (Doistributed Energy Resorces), con otros controladores en tiempo real, para lo cual se requiere tener un único identificador para cada uno de los controladores y su DER local.
- **Concurrencia:** Cada controlador debe tener un estado actualizado a la fecha del sistema completo, especialmente para las entradas al algoritmo del control de la microrred corriendo en cada controlador. Esto es un requerimiento clave para proteger la integridad del sistema de ser violado, de otra manera,podrian surgir salidas inconsistentes del algoritmo y comandos de control, las cuales podrian llevar a perturbaciones en la red.
- **Escalabilidad:** El sistema debe permitir escalar agregando o quitando componentes de potencia sin que se afecte la operación se tenga que re-ingenierizar el algoritmo de control.
- **Tolerancia a las fallas:** El sistema debe mantenerse disponible y operando a minimo nivel de confiabilidad. Esto también incluye el proceso de recuperación en caso de fallas y posible redundancia que podría aumentar la confiabilidad de la microrred.

En la Figura 2.13 se representa un sistema que carece de MGCC (*Microgrid Central Controller* o bien en español, Microcontrolador Centralizado de Microrred), reemplazandolo por controladores locales con una alta capacidad de controlador descentralizado de la microred (DMGC).

Figura 2.12: Arquitectura de Control Descentralizada para Microrred Fuente: [5]

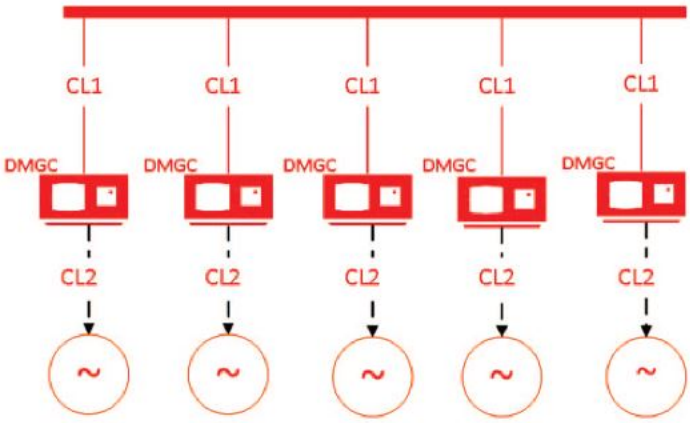
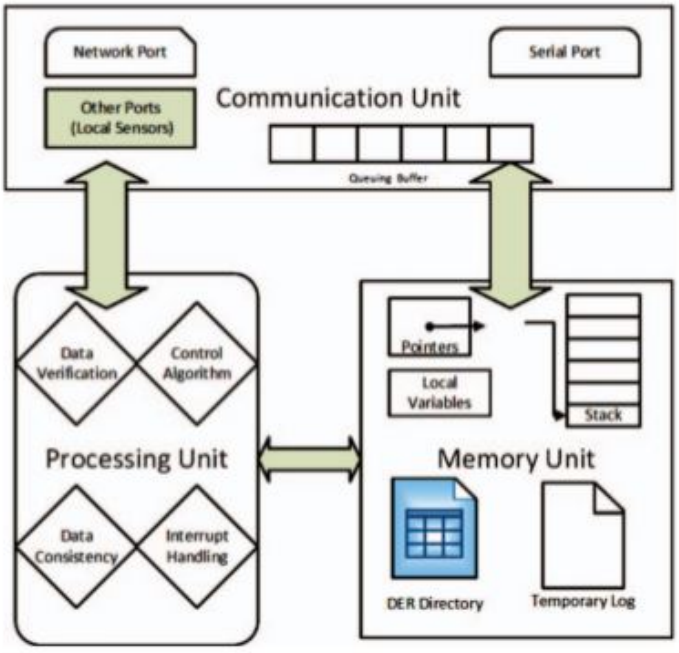


Figura 2.13: Diseño de Control Descentralizado Fuente: [5]

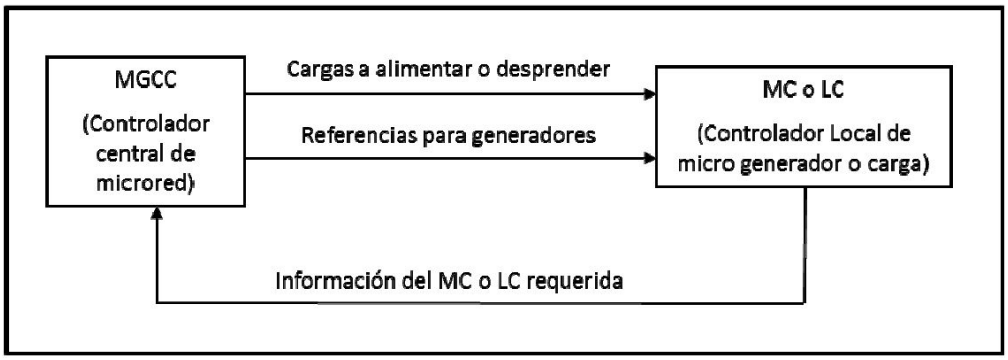


2.4.4. Control centralizado

El control centralizado, a diferencia del descentralizado, se realiza a través de un operador central el cual coordina el funcionamiento de las unidades.

A través de un control centralizado el MGCC optimiza el intercambio de potencia con el sistema, maximizando la producción local la cual es función de los precios de mercado y las restricciones de seguridad. Esto se logra enviando referencias las unidades de generación y a las cargas controlables de la microrred. En la Figura 4 se muestra un ejemplo de intercambio de información cuando se emplea una estrategia de control centralizado, y se indica la importancia de la comunicación entre el MGCC y el LC o MC. El MGCC toma decisiones cada ciertos intervalos pre-establecidos de tiempo, los cuales pueden ser desde algunos minutos hasta días.

Figura 2.14: Ejemplo de Control Centralizado de Microrred, Fuente: [6]



El MGCC debe considerar lo siguiente:

- Restricciones de seguridad de la red.
- Predicciones de demanda y recursos renovables
- Usando un proceso de optimización determina:
  - Referencias de las unidades GD.
  - Referencias de las cargas.

Según las señales del MGCC, los LC ajustan la generación y los niveles de demanda para presentar sus ofertas en el siguiente periodo. Las funciones que se pueden implementar para llevar a cabo el control centralizado de una Microrred incluyen generación de energía, carga, seguridad despacho económico y previsión sobre el compromiso de la unidad [6].

Arquitectura Centralizada

En las arquitecturas centralizadas, la relación entre los componentes sigue un patrón muy característico, en el que hay una jerarquía definida de manera tal que ciertos componentes requieren información o servicios que otros ofrecen.

El modelo centralizado es el que sido ampliamente utilizado en los Sistemas de Información de las grandes organizaciones en décadas anteriores, mediante un Host que ejecutaba el 100 % de la lógica del sistema, residiendo únicamente en el terminal de usuario las funciones de presentación. A este tipo de aplicaciones, que concentran todas las lógicas funcionales del software (presentación, negocio y acceso a datos) en un mismo componente se les denomina *aplicaciones monolíticas*.

En la Figura 2.10 a través de un control centralizado el MGCC optimiza el intercambio de potencia con el sistema, maximizando la producción local la cual es función de los precios de mercado y las restricciones de seguridad. Esto se logra enviando referencias las unidades de generación y a las cargas controlables de la microrred.

2.5. Lenguajes de Programación

Un lenguaje de programación es un idioma artificial diseñado para expresar instrucciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.

Está formado de un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila y se mantiene el código fuente de un programa informático se le llama programación.

También la palabra programación se define como el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos, a través de los siguientes pasos:

- El desarrollo lógico del programa para resolver un problema en particular.
- Escritura de la lógica del programa empleando un lenguaje de programación específico (codificación del programa)
- Ensamblaje o compilación del programa hasta convertirlo en lenguaje de máquina.
- Prueba y depuración del programa.
- Desarrollo de la documentación.

Existe un error común que trata por sinónimos los términos 'lenguaje de programación' y 'lenguaje informático'. Los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como por ejemplo el HTML. (lenguaje para el marcado de páginas web que no es propiamente un lenguaje de programación sino un conjunto de instrucciones que permiten diseñar el contenido y el texto de los documentos)

Permite especificar de manera precisa sobre qué datos debe operar una computadora, cómo deben ser almacenados o transmitidos y qué acciones debe tomar bajo una variada gama de circunstancias. Todo esto, a través de un lenguaje que intenta estar relativamente próximo al lenguaje humano o natural, tal como sucede con el lenguaje Léxico. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción del programa de forma colaborativa.

A continuación se presentan los principales lenguajes de programación utilizados en la actualidad en sistemas. En particular, el programa SCADA de Huatacondo se encuentra programado en C-Sharp, el cual es un lenguaje de programación orientado a objetos.

2.5.1. Lenguajes de Programación Orientado a Objetos (POO)

La programación orientada a objetos es una paradigma en el cual los .ºbjetos.ºperan los datos de entrada para obtener datos de salida específicos, donde cada objeto ofrece una funcionalidad especial. Pensar en términos de objetos es muy similar a "modelarün objeto añadiendole características además de algunas funcionalidades. Dos ejemplos pueden ser:

- Un vehículo es el objeto; su color, modelo o marca sus caracterísitcas; sus funcionalidades pueden ser ponerse en marcha, parar o estacionarse.
- Una fracción es el objeto; su numerador y su denominador sus características; sus métodos pueden ser simplificarse, amplificarse, sumarse con otra fracción o número, restarse con otra fracción, etc.

Estos objetos se pueden utilizar en programas para resolver problemas. Por ejemplo un programa de matemática puede hacer uso de objetos fracción y un programa que gestione un taller de autos utilizará objetos coches. Los programas orientados a objetos utilizan otros objetos para realizar las acciones que desean realizar y ellos mismos también son objetos, es decir, el taller de coches será un objeto que utilizará objetos coche, herramienta, mecánico, etc.

Clases en POO

Una clase es una declaracipon de objetos: cuando se programa un objeto y se definen sus características y funcionalidades lo que se está haciendo en realidad es programar una clase, que es así mismo una abstracción de objetos.

Propiedades en clases Las propiedades o atributos son las características que definen a los objetos. Al definir una propiedad se especifica su nombre y su tipo, de esta manera, estas propiedades son "variables"donde se almacenan los datos asociados a los objetos.

Métodos en Clases Son las funcionalidades que se asocian a los objetos y son llamadas métodos. Estos métodos son las funciones que están asociadas a un objeto.

Objetos de POO

Los objetos mismos son ejemplares de una clase cualquiera. Al crear un objeto, se debe especificar la clase a partir de la cual se creará; a esta acción de crear un objeto a partir de una clase se le llama ïnstanciar". Luego, se le llamará clase al concepto o definición que se dé del objeto mientras que cuando se hable del objeto en concentro se le llamará por su nombre.

Para crear un objeto se debe escribir una instrucción especial que puede ser distinta dependiendo del lenguaje de programación empleado . A continuación se muestra el siguiente

ejemplo con un objeto Auto.

miAuto= new Auto()

Con la palabra "new"se especifica que se tiene que crear una instancia de la clase que sigue a continuación. Dentro de los paréntesis podrían necesitar colocarse parámetros con lo cual iniciar el objeto de la clase auto.

Estados en Objetos Cuando se tiene un objeto, sus propiedades toman valores. Este valor en concreto se llama .estado". Para acceder a un estado de un objeto, ver sus valor o cambiarlo, se utiliza el operador punto.

miAuto.color=rojo

El objeto es miAuto, luego se coloca el operador punto y por úlimo el nombre de la propiedad a la que se desea acceder. En el ejemplo anterior se cambia el valor del estado de la propiedad del objeto a rojoçon una asignación simple.

C++

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. ... Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma.

C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

El nombre *C Sharp* fue inspirado por el signo '' que se compone de dos signos '+' pegados.

Aunque C forma parte de la plataforma .NET, ésta es una API, mientras que C es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows Microsoft, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

Métodos en C #

- **Handlers:** Un "Handler" es un método controlador de eventos en el receptor de eventos, es decir, el que se ejecuta en el momento de que el receptor es notificado de que ha ocurrido un evento. Para esto, se debe relacionar el método con el evento que va a controlar cuando ocurra.
- **Timer:** juega un rol importante en el desarrollo de programas desde ambos lados, Cliente y Servidor, tanto como en los Servicios de Windows.Con el *Timer Control* podemos plantear eventos en intervalos específicos de tiempo sin la necesidad de interacción de otro hilo o proceso.

2.5.2. Modularización

Modularizar significa dividir un problema en partes funcionalmente independientes, que encapsulen operaciones y datos. No se trata simplemente de subdividir el código de un sistema de software en bloques con un número de instrucciones dado. Separar en funciones lógicas con datos propios y datos de comunicación perfectamente especificados. Lo anterior es la base para modularizar el problema que se aborda en la presente memoria.

Abstracción La descomposición tiene siempre un objetivo. Se busca obtener:

- Alta Cohesión: medida del grado de identificación de un módulo con una función concreta
- Bajo Acoplamiento: medida de la interacción de los módulos que constituyen un programa.

Cuando se descompone un problema en subproblemas, deben ser de forma tal que:

- Cada subproblema está en un mismo nivel de detalle.
- Cada subproblema puede resolverse lo más independientemente posible.
- Las soluciones de los subproblemas puede combinarse para resolver el problema original.

Descomposición - ¿Qué son los Módulos?

Es un conjunto de instrucciones que cumplen una tarea específica bien definida, se comunican entre sí adecuadamente y cooperan para conseguir un objetivo común.

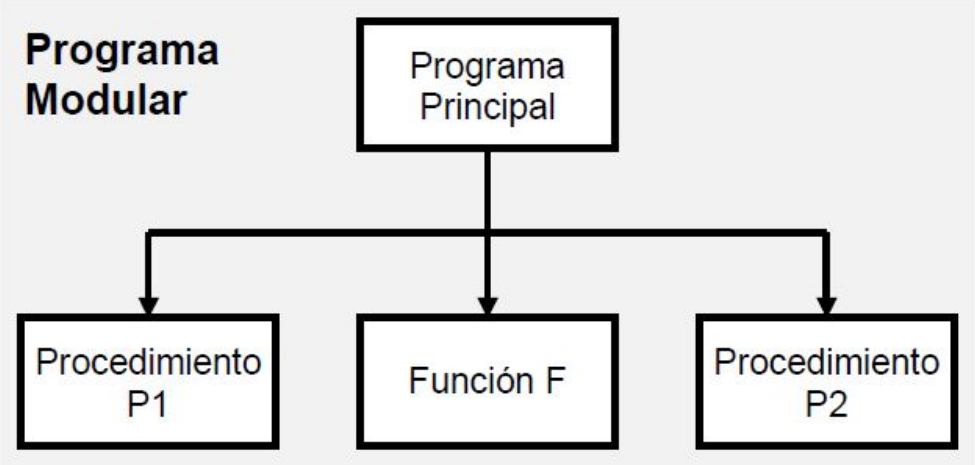
- Se descompone de problemas grandes a problemas pequeños
- Cada módulo encapsula, acciones tareas o funciones
- Hay que representar los objetos relevantes del problema a resolver.

De esta manera, se puede diseñar un programa de la manera que se muestra en la Fig. 2.15.

Diseño Top-Down

Cuando se escriben programas de un tamaño y complejidad moderados, se encuentra una gran dificultad para abarcar todo el programa de una sola vez. La filosofía del diseño top-down consiste en llevar a cabo una tarea mediante pasos sucesivos a un nivel de detalle cada vez más bajo. Para ello sería necesario dividir un programa en diferentes módulos procedimientos, funciones y otros bloque de código. El diseño top-down es una de las metodologías más empleadas en programación. Está basada en la técnica de resolución humana de problemas: divide y vencerás. Consiste en dividir el algoritmo en unidades más pequeñas sucesivamente hasta que sean directamente ejecutables en el ordenador.

Figura 2.15: Diseño de un Programa Modular Fuente: [7]

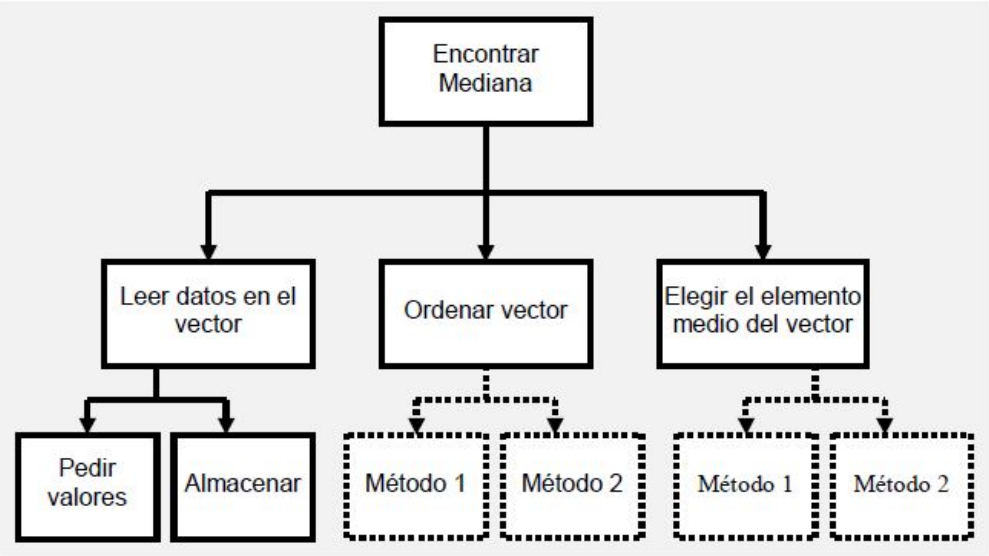


Normalmente, los componentes (P1, F, P2) (ver Figura 2.15)son bastante independientes del algoritmo principal y podrían ser diseñados sin considerar el contexto. Estos componentes reciben el nombre de **módulos**: "un algoritmo autocontenido, que puede ser diseñado independientemente del contexto en el que va a ser usado."

Herramientas: **procedimientos** y **funciones** y, según el lenguaje, módulos (ej., Módula-2).

Ejemplo: supongamos que queremos encontrar la mediana de una colección de datos. Al principio, cada módulo es poco más que una sentencia de qué se necesita resolver y se evitan los detalles de cómo hacerlo. Los módulos se van refinando en pasos sucesivos que resuelven problemas cada vez más pequeños y concretos y contienen más detalles acerca de cómo resolver el problema. El proceso de refinamiento continúa hasta que los módulos en la parte baja de la jerarquía son suficientemente simples como para ser traducidos a un lenguaje de programación. Un programa de este tipo se dice que es modular. Los módulos diseñados son independientes estre sí, excepto en la **interfaz** que los comunica. A continuación en la Figura 2.16 se muestra el ejemplo de los datos y el procedimiento abstracto de cómo calcular su mediana.

Figura 2.16: Ejemplo de Diseño de un Programa Modular para calcular la mediana de un conjunto de números Fuente: [7]



2.5.3. Diagramas UML

El Lenguaje Unificado de Modelado, mejor conocido como UML, es un conjunto de herramientas de modelado que orienta la creación y notación de muchos tipos de diagramas, incluidos los diagramas de comportamiento, los diagramas de interacción y los diagramas de estructuras. En este trabajo se utilizará el Diagrama UML del sotfware para identificar oportunidades de mejora que permitan perfeccionar el funcionamiento del sistema SCADA.

Diagramas de secuencia

Un diagrama de secuencia es un tipo de diagrama de interacción porque describe cómo —y en qué orden— un grupo de objetos funcionan en conjunto. Tanto los desarrolladores de software como los profesionales de negocios usan estos diagramas para comprender los requisitos de un sistema nuevo o documentar un proceso existente. A los diagramas de secuencia en ocasiones se los conoce como diagramas de eventos o escenarios de eventos.Observa que

hay dos tipos de diagramas de secuencia: los diagramas UML y los diagramas que se basan en código. Los últimos se obtienen de un código de programación y no serán cubiertos en esta guía. El software de diagramas UML de Lucidchart está equipado con todas las figuras y funciones que necesitarás para modelar ambos.

Los beneficios de los diagramas de secuencia Los diagramas de secuencia pueden ser referencias útiles para las empresas y otras organizaciones. Prueba dibujar un diagrama de secuencia para:

- Representa los detalles de un caso de uso en UML.
- Representa los detalles de un caso de uso en UML.
- Modelar la lógica de una operación, una función o un procedimiento sofisticados.
- Modelar la lógica de una operación, una función o un procedimiento sofisticados.
- Ve cómo los objetos y los componentes interactúan entre sí para completar un proceso.
- Planificar y comprender la funcionalidad detallada de un escenario actual o futuro.

Los casos de uso para los diagramas de secuencia Los siguientes escenarios son ideales para usar un diagrama de secuencia:

- **Escenario de uso:** Un escenario de uso es un diagrama de cómo se podría usar potencialmente tu sistema. Es una excelente manera de asegurar que has estudiado la lógica de cada escenario de uso para el sistema.
- **Lógica del método:** Al igual que utilizarías un diagrama de secuencia UML para explorar la lógica de un caso de uso, puedes usarlo para explorar la lógica de cualquier función, procedimiento o proceso complejo.
- **Lógica de servicio:** Si consideras que un servicio es un método de alto nivel empleado por diferentes clientes, un diagrama de secuencia es una forma ideal de trazarlo.

Uno de los programas que se utilizan mucho para realizar Diagrama de secuencia, es el Microsoft Visio. Este programa trabaja con archivos de extensiones .vsd y .vdx.

2.6. Estándar y Servidores OPC

2.6.1. Estándar OPC

El OPC (*OLE for Process Control*) es un estándar de comunicación en el campo del control y supervisión de procesos industriales, basado en una tecnología Microsoft, que ofrece una interfaz común para comunicación que permite que componentes de software individuales interactúen y compartan datos.

La comunicación OPC se realiza a través de una arquitectura Cliente-servidor. El servidor OPC es la fuente de datos (como un dispositivo hardware a nivel de planta) y cualquier aplicación basada en OPC puede acceder a dicho servidor para leer/escribir cualquier variable que ofrezca el servidor.

Es una solución abierta y flexible al clásico problema de los drivers propietarios. Prácticamente todos los mayores fabricantes de sistemas de control, instrumentación y de procesos han incluido OPC en sus productos.

2.6.2. ¿Qué es un Servidor OPC?

Un servidor OPC es una aplicación de software (*driver*) que cumple con una o más especificaciones definidas por la OPC Foundation. El Servidor OPC hace de interfaz comunicando por un lado con una o más fuentes de datos utilizando sus protocolo nativos (típicamente PLCs, DCSs, básculas, Modulos I/O, controladores, etc.) y por el otro lado con Clientes OPC (típicamente SCADAs HMIs, generadores de informes, generadores de gráficos, aplicaciones de cálculos, etc.). En una arquitectura Cliente OPC/ Servidor OPC, el Servidor OPC es el esclavo mientras que el Cliente OPC es el maestro. Las comunicaciones entre el Cliente OPC y el Servidor OPC son bidireccionales, lo que significa que los Clientes pueden leer y escribir en los dispositivos a través del Servidor OPC.

Existen cuatro tipos de servidores OPC definidos por la OPC Foundation, y son los siguientes:

- Servidor OPC DA – Basado en Spezifikationsbasis: OPC Data Access - especialmente diseñado para la transmisión de datos en tiempo real.
- Servidor OPC HDA– Basado en la especificación de Acceso a Datos Historizados que provee al Cliente OPC HDA de datos históricos.
- Servidor OPC AE Server– Basado en la especificación de Alarmas y Eventos – transfiere Alarmas y Eventos desde el dispositivo hacia el Cliente OPC AE.
- Servidor OPC UA – Basado en la especificación de Arquitectura Unificada – basado en el set mas nuevo y avanzado de la OPC Foundation, permite a los Servidores OPC trabajar con cualquier tipo de datos.

En conjunto, los tres primeros tipos de Servidores OPC se conocen como Servidores OPC Çlásicos"para distinguirlos de OPC UA que se convertirá en la base de las futuras arquitecturas OPC.

Arquitectura de Servidor OPC Comunicaciones Cliente OPC / Servidor OPC (Servidor OPC DA, Servidor OPC HDA, Servidor OPC AE)

Los Servidores OPC clásicos utilizan la infraestructura COM/DCOM de Microsoft Windows para el intercambio de datos. Lo que significa que esos Servidores OPC deben instalarse bajo el Sistema Operativo de Microsoft Windows. Un Servidor OPC puede soportar comunicaciones con múltiples Clientes OPC simultáneamente.

Servidor OPC - Traducción de Datos/Mapping

La principal función de un Servidor OPC es el traducir datos nativos de la fuente de datos en un formato OPC que sea compatible con una o más especificaciones OPC mencionadas anteriormente (ejemplo: OPC DA para datos en tiempo real). Las especificaciones de la OPC Foundation solo definen la porción OPC de las comunicaciones del Servidor OPC, así que la eficiencia y calidad de traducción del protocolo nativo a OPC y de OPC al protocolo nativo dependen enteramente de la implementación del desarrollador del Servidor OPC.

Servidor OPC –Comunicación Fuente de Datos

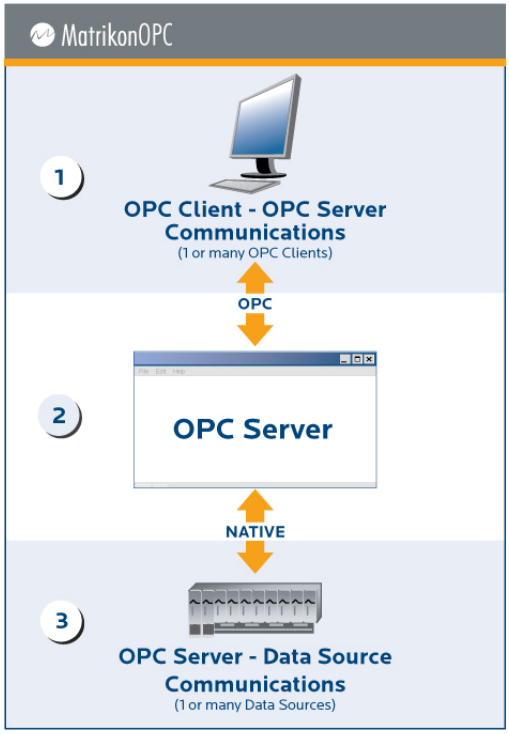
Los Servidores OPC comunican nativamente con las fuentes de datos, por ejemplo: dispositivos, controladores y aplicaciones. Las especificaciones de la OPC Foundation no especifican como el Servidor OPC se debe comunicar con la fuente de datos porque hay una gran variedad de fuentes de datos disponibles en el mercado. Cada PLC, DCS, controlador, etc. tiene su propio protocolo de comunicación o API que a su vez permiten la utilización cualquier cantidad de conexiones físicas (serial RS485 o RS232, Ethernet, wireless, redes propietarias, etc.).

Dos ejemplos comunes de cómo se comunican los Servidores OPC con la Fuente de Datos son:

- A través de una interfaz de programación de aplicaciones (API) para un driver personalizado escrito específicamente para la Fuente de Datos.
- A través de un protocolo que puede o no ser propietario, o basado en un estándar abierto (por ejemplo utilizando el protocolo Modbus. (MatrikonOPC Server para Modbus)

En la Figura 2.6.2 se muestra un esquema genérico del funcionamiento de los servidores OPC, con uno de los cuáles opera el SCADA de Huatacondo. En este esquema se muestra como se conectan el Cliente con el Servidor OPC y luego con la Fuente de Datos.

Figura 2.17: Esquema de Servidor OPC



2.7. Frontend y Backend

Frontend es la parte de un sitio web o aplicación que interactúa con los usuarios, por eso decimos que está del lado del cliente. Backend es la parte que se conecta con la base de datos y el servidor que utiliza dicho sitio web o aplicación, por eso decimos que el backend corre del lado del servidor. Estos dos conceptos explican a grandes rasgos cómo funciona una página web, aplicación o programa.

2.7.1. Frontend

Frontend es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son también todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios.

HTML, CSS y JavaScript son los lenguajes principales del Frontend, de los que se desprenden una cantidad de frameworks y librerías que expanden sus capacidades para crear cualquier tipo de interfaces de usuarios. React, Redux, Angular, Bootstrap, Foundation, LESS, Sass, Stylus y PostCSS son algunos de ellos, sin perjuicio de poder programar el Frontend en otros lenguajes también.

2.7.2. Backend

Backend es la capa de acceso a datos de un software o cualquier dispositivo, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. El Backend también accede al servidor, que es una aplicación especializada que entiende la forma como el navegador solicita cosas.

Algunos de los lenguajes de programación de Backend son Python, PHP, Ruby, C# y Java, y así como en Frontend, cada uno de los anteriores tiene diferentes frameworks que te permiten trabajar mejor según el proyecto que estás desarrollando. En Platzi tenemos Django, Laravel, Ruby On Rails y ASP.Net, los hemos elegido sobre todo porque tienen una gran comunidad que los respalda.

2.7.3. Métricas de desemmpeño

Elapsed Inclusive values

El tiempo total utilizado ejecutando una función y sus funciones secundarias.

Los valores de *Elapsed Inclusive* incluyen los intervalos que fueron utilizados de la ejecución directa del código y los intervalos que fueron utilizados ejecutando las las funciones secundarias de la función objetivo. Los intervalos de ejecución de la función o de sus funciones secundarias que incluyen esperar al sistema operativo también se incluyen en los *Elapsed Inclusive values*.

Elapsed Exclusive values

El tiempo total utilizado ejecutando una función, exluyendo el tiempo utilizado en sus funciones secundarias.

Elapsed Exclusive values incluye los intervalos que fueron utilizados ejecutando directamente el código de la función, independientemente si se produjeron eventos del sistema operativo en el intervalo. Todos los intervalos utilizados en funciones secundarias que fueron llamadas por la función objetivo no están incluidas en Elapsed Exclusive values.

Número de llamados

Son la cantidad de llamados que se hacen a las funciones o a los módulos del programa en cuestión.

Capítulo 3

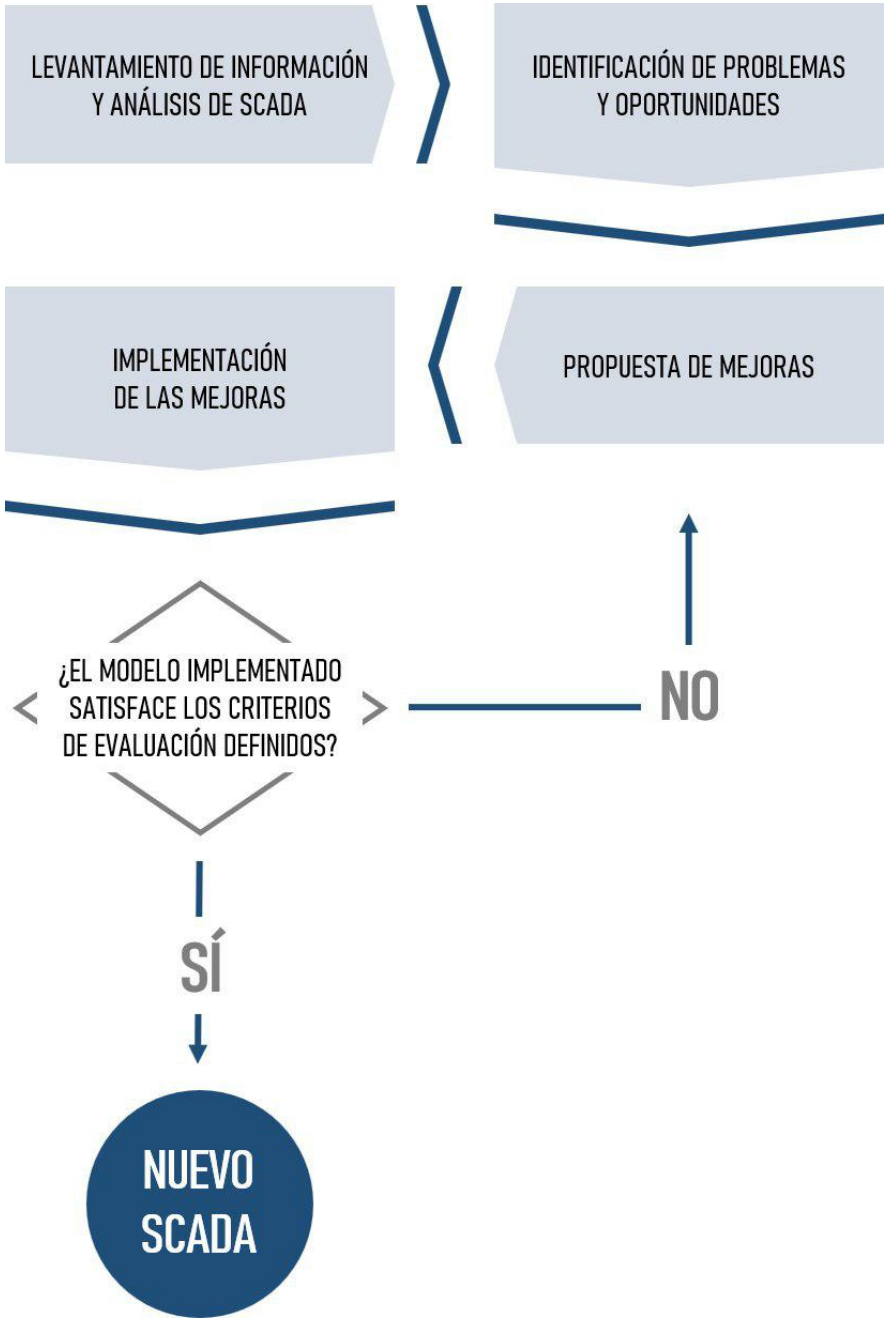
Propuesta Metodológica

En este capítulo se explicarán los pasos que se seguirán para realizar el trabajo propuesto del rediseño del software SCADA de Huatacondo mediante la modularización del código.

La propuesta metodológica sigue los siguientes pasos, los cuales aparecen en forma de flujo en la Figura 3.1 y se desglosan a continuación en pasos generales:

- **Levantamiento de Información y Análisis del Sistema SCADA:**
  - Rescatar códigos
  - Generar UML a partir de los códigos del programa
  - Analizar los UML de los códigos
- **Identificación de Problema y Oportunidades:**
  - Desglosar las funciones que operan en el SCADA
  - Clasificar funciones
  - Reconocer los roles que cumplen las funciones
  - Identificar oportunidades (funciones repetidas, objetos repetidos, etc.)
- **Propuesta de Mejoras:**
  - Proponer módulos concretos
  - Nuevo código
  - Nuevas estructuras
- **Implementación de Mejoras:**
  - Trabajo sobre el código base
  - Depuración, pruebas y experimentación
  - Métricas de desempeño

Figura 3.1: Propuesta Metodológica, Fuente: Realización personal



3.1. Levantamiento de Información y Análisis de Sistema SCADA

Este apartado contiene todo lo referente a la etapa inicial de exploración del código de programación del SCADA en el caso particular de Huatacondo. Aquí se expondrá el rescate del código, los UML generados a partir del programa y su respectivo posterior análisis.

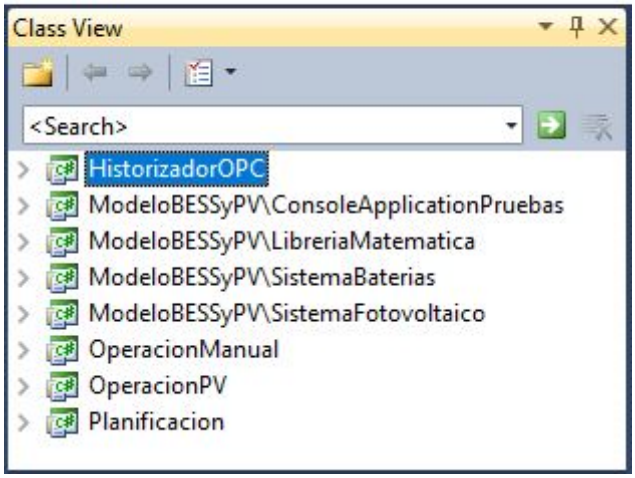
3.1.1. Rescatar Código Fuente

En primera instancia, se necesita rescatar el código fuente del programa que se modularizará. En particular, se tomó la carpeta proporcionada por Fernando Lanas[1] quien actualmente trabaja con el sistema SCADA en cuestión.

En el esquema actual, el programa se encuentra operado por 8 clases/módulos principales, tal como se muestra en la Figura 3.2. Estos módulos en los cuales se divide el programa son los siguientes:

- Historizador OPC
- Modelo BESS y PV / ConsoleApplicationPruebas
- Modelo BESS y PV / LibreríaMatemática
- Modelo BESS y PV / Sistema de Baterías
- Modelo BESS y PV / Sistema Fotovoltaico
- OperacionManual
- OperaciónPV
- Planificación

Figura 3.2: Vista de Clases en Visual Studio, Fuente: SCADA operando en Huatacondo



3.1.2. Generación de Diagrama UML

En este apartado se mostará el Diagrama UML generado a partir del programa SCADA de Huatacondo. Se podrá ver que el diagrama aparece como un árbol en el cuál las funciones se concentran principalmente en el documento que lleva por nombre "form1". Este documento (siguiendo la estructura del programa), es el que se encarga de la interacción con el usuario y por tanto, correspondería al HMI del programa. Sin perjuicio de lo anterior, el documento "form1çontiene funciones que no tienen que ver con las funciones de un HMI, por ejemplo, funciones que realiza o no el *módulo diesel*.

A continuación se muestra la Fig. 3.4 donde aparece el árbol del diagrama UML de clases. Además, en la imagen anterior, se presenta un acercamiento a la raíz"de donde nace el diagrama UML, particularmente donde se muestra la caja del *"form1"* saliendo desde *"Program >Main"* en la Fig. 3.3.

Figura 3.3: Acercamiento de Diagrama UML - SCADA de Microrred Huatacondo, Fuente: SCADA operando en Huatacondo

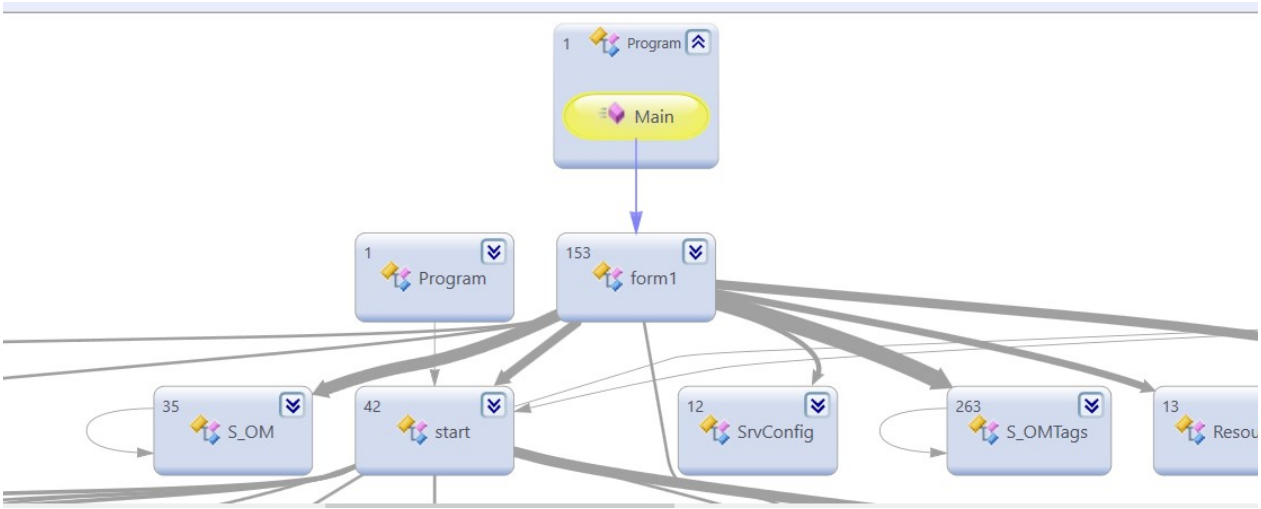
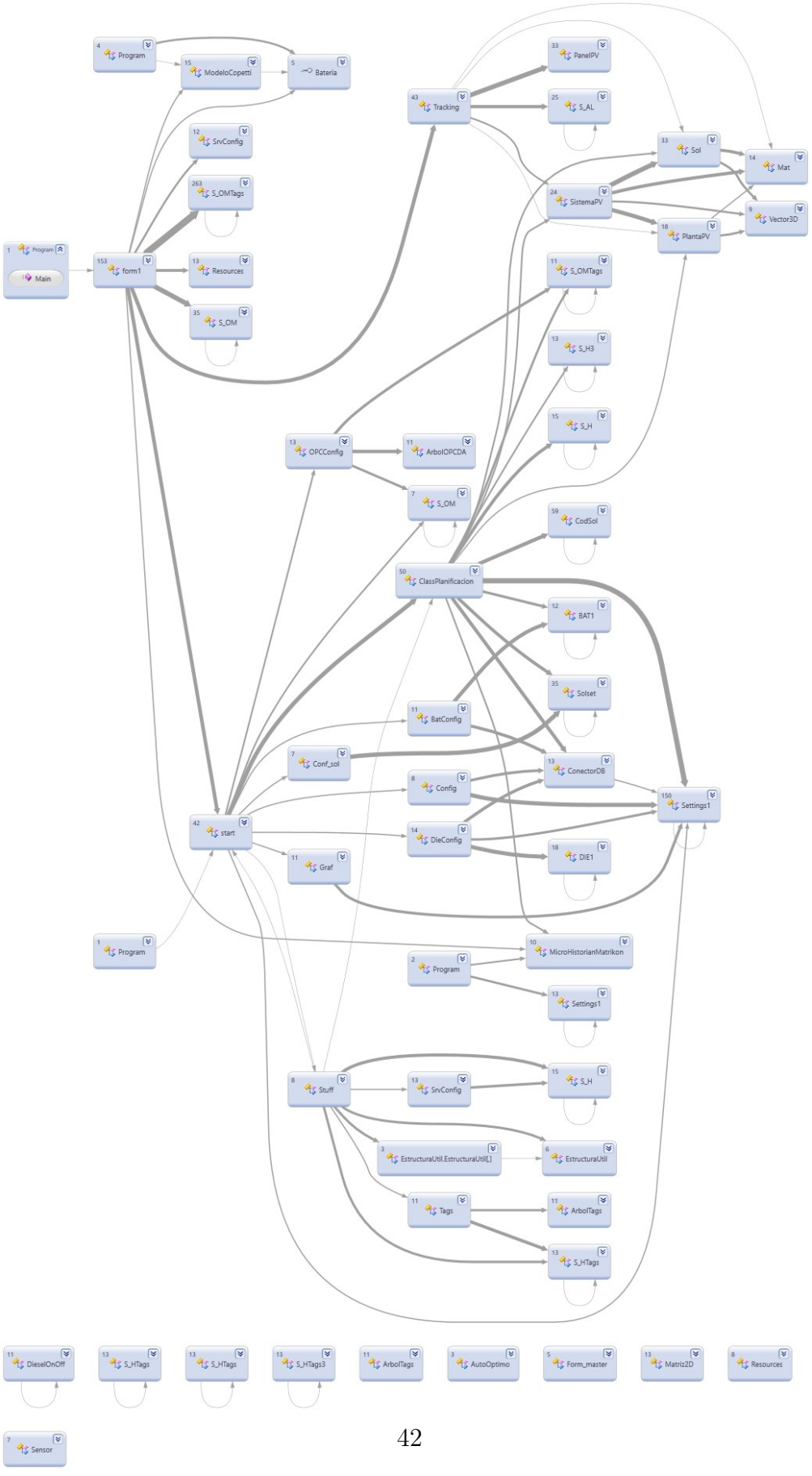


Figura 3.4: Diagrama UML - SCADA de Microrred Huatacondo, Fuente: SCADA operando en Huatacondo

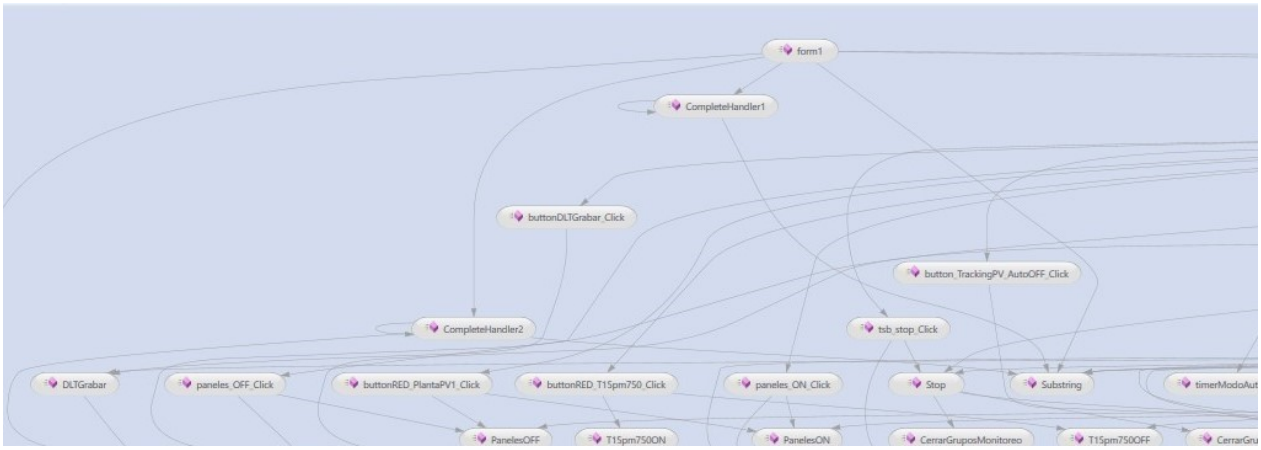


3.1.3. Análisis de Diagrama UML

En primera instancia, se crea el bloque *"Program.cs"*, cuyo código aparece en el Apéndice A en la Fig. A.1, en el cual se muestra la forma del constructor que da inicio al *"form1"*. Para muchas aplicaciones debe existir un punto de partida y .NET refiere a la clase del programa con la cual será creada automáticamente para la aplicación de formularios. **Application. Run(new object())** inicializará el objeto correspondiente para la primera ejecución.

Una vez creada la *form1()* por esta instancia, se hace click en *form1()* y despliega muchas funciones distintas como se muestra en la Fig. 3.6. En este paso, lo importante es entender el programa desde el form1(), el cual debiese ser como se mencionó anteriormente, el código que cumple la función de HMI.

Figura 3.5: Diagrama UML - Acercamiento al *form1()*, Fuente: SCADA operando en Huatacondo

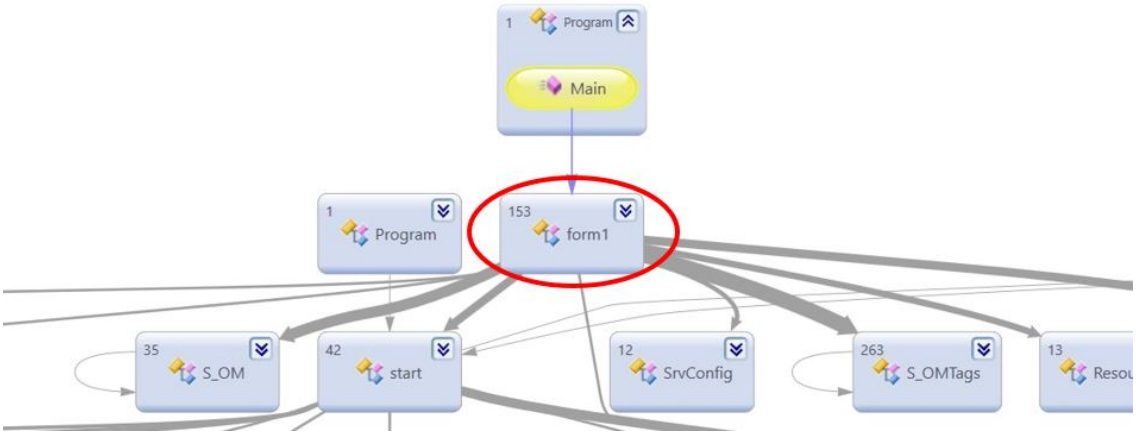


En la Fig. 3.5 se muestran las relaciones del form1 con otras funciones. Estas flechas salen hacia:

- InitializeComponent
- FallaOPCserver
- CompleteHandler1
- CompleteHandler2
- Substring
- AgregarEvento

Estas funciones se encuentran todas dentro de la misma clase Form1(). Además desde el Main sale una flecha a conectar Form1(), donde ese Main viene siendo la clase Operacion-Manual cuyo código se muestra en el Apéndice en la Fig. A.1.

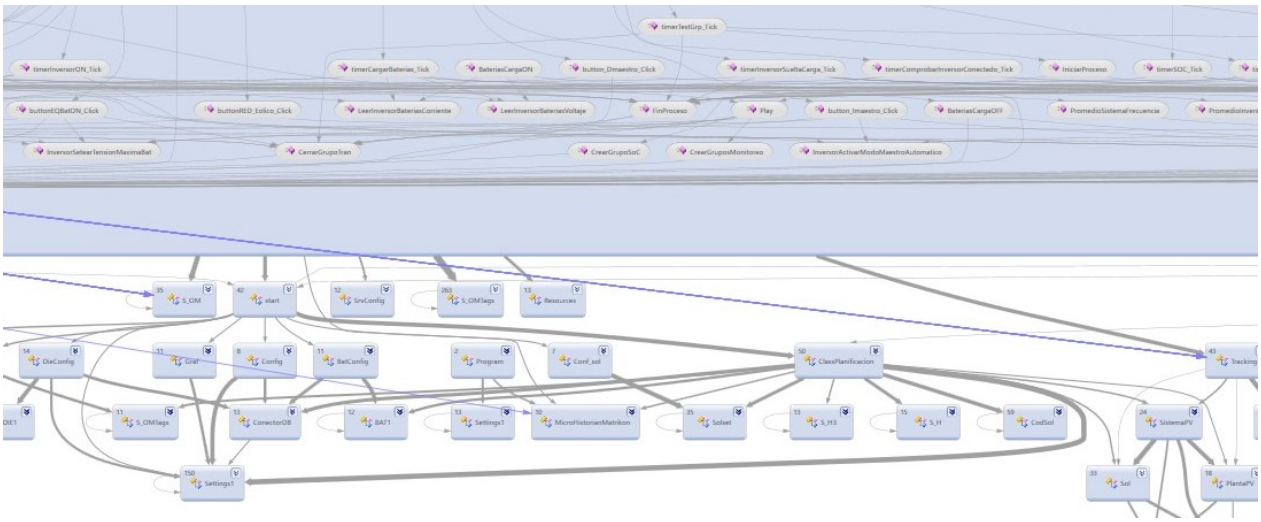
Figura 3.6: Diagrama UML - Form1, Fuente: SCADA operando en Huatacondo



Además de haber relaciones dentro de la misma clase, también salen flechas desde *form1()* hacia los bloques externos al form1(), tal como se ve en la Fig. 3.7:

- (15) ModeloCopetti
- (35) S_OM
- (10) MicroHistorianMatrikon
- (43) Tracking

Figura 3.7: Diagrama UML - Form1, nexos con otros Bloques, Fuente: SCADA operando en Huatacondo



Análisis del Código del *form1()*

A continuación se realizará un análisis del código perteneciente al documento form1(), cuyo funcionamiento se estudiará parte por parte para proceder al segundo paso del trabajo: la identificación de problemas y oportunidades.

En la Fig. 3.9 se puede ver el código de la clase Form1() observándose que está dividida en 4 partes principales: *Variables, Propiedades, Constructor y Métodos*. Una vez identificada esta estructura se deben analizar las funciones presentes en cada una e identificar sus funciones y llamados.

Figura 3.8: Diagrama UML - Código declaración de Métodos y Funciones, Fuente: SCADA operando en Huatacondo

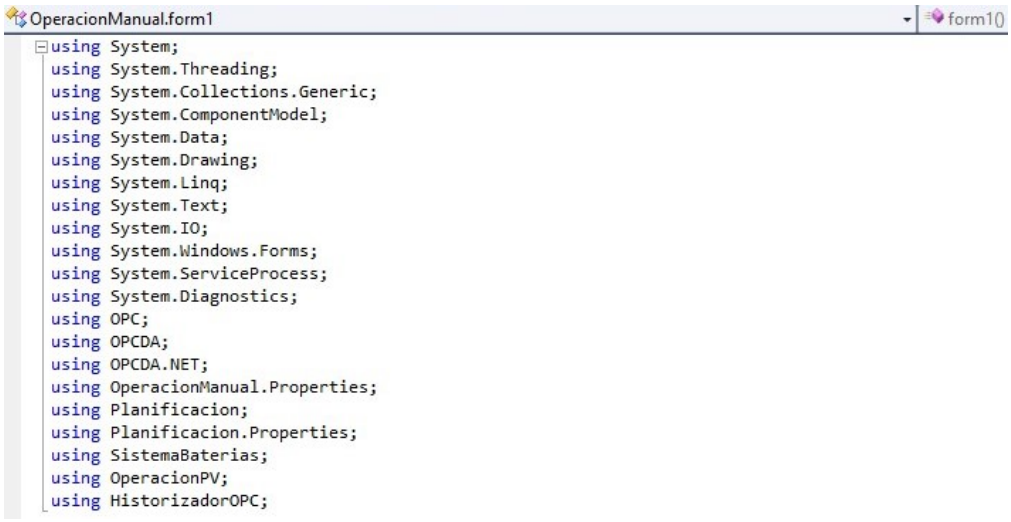
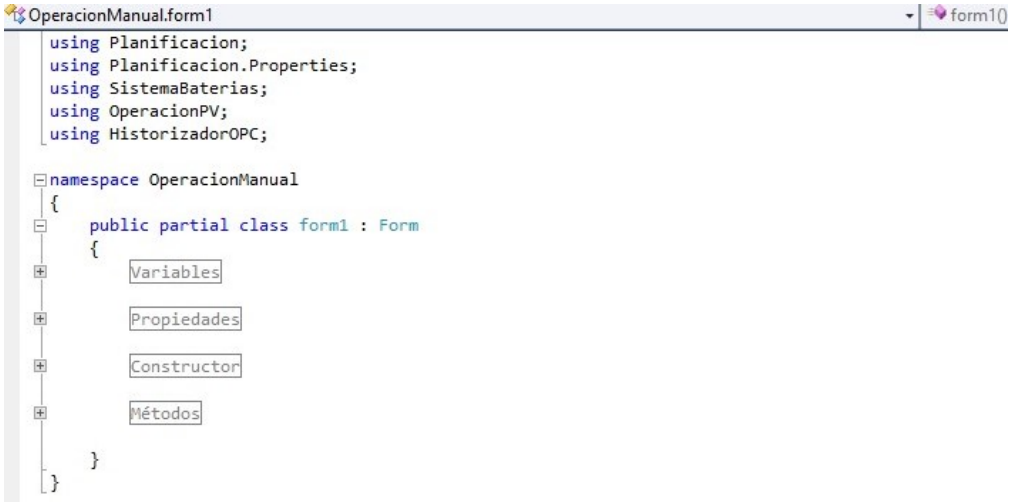


Figura 3.9: Diagrama UML - Código Estructura form1(), Fuente: SCADA operando en Huatacondo



Variables

A continuación se muestra la declaración de *variables* en el código del *form1()* en la Fig. 3.10

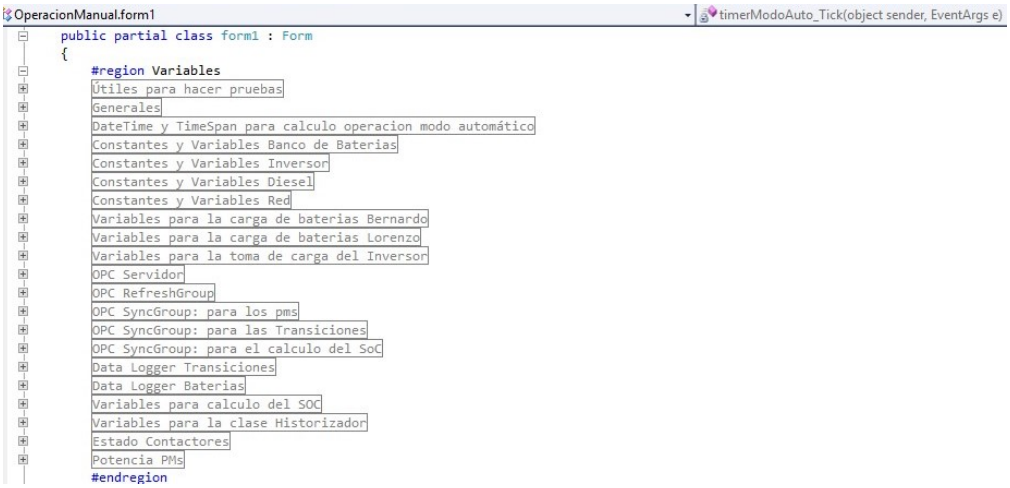


Figura 3.10: Código Estructura form1()- Variables, Fuente: SCADA operando en Huatacondo

En este apartado se puede chequear que existen variables tanto de cálculo de para el inversor, diesel, banco de baterías y la red; variables para controlar el OPC del servidor, el RefreshGroup (transiciones, pms y SOC); además de variables para el estado de los contactores y la potencia de los PMs. Además contiene variables de tiempo como DateTime y TimeSpan para monitorear directamente el funcionamiento del programa.

Propiedades

A continuación se muestra la declaración de *propiedades* en el código del *form1()* en la Fig. 3.11

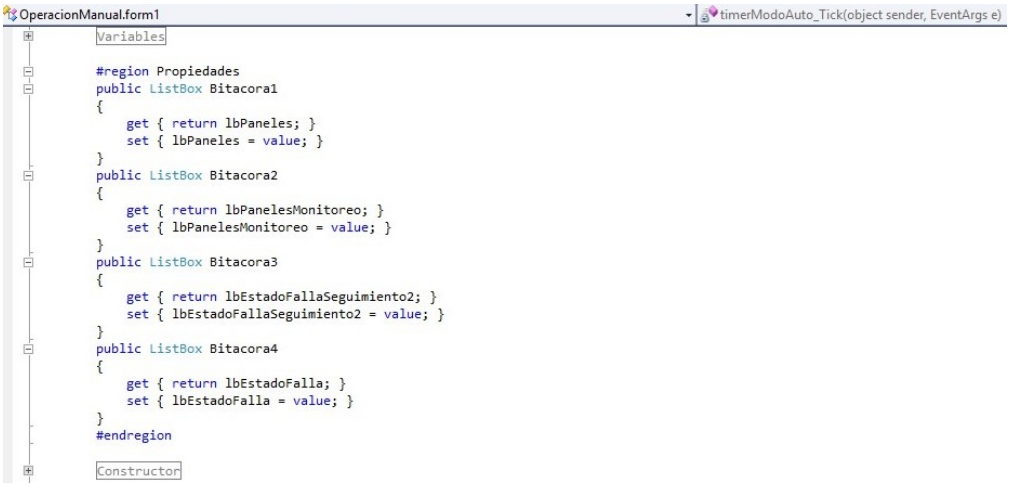


Figura 3.11: Código Estructura form1()- Propiedades, Fuente: SCADA operando en Huatacondo

En este apartado se puede apreciar que la estructura del apartado *Propiedades* se construye en base a clases públicas de la función *ListBox*la cual maneja la lectura y el *"setting"* de 4 funciones que despliegan datos en la ventana del programa ESUSCON:

- lbPaneles
- lbPanelesMonitoreo
- lbEstadoFallaSeguimiento2
- lbEstadoFalla

Constructor

A continuación se muestra la declaración de *construcción* en el código del *form1()* en la Fig. 3.12

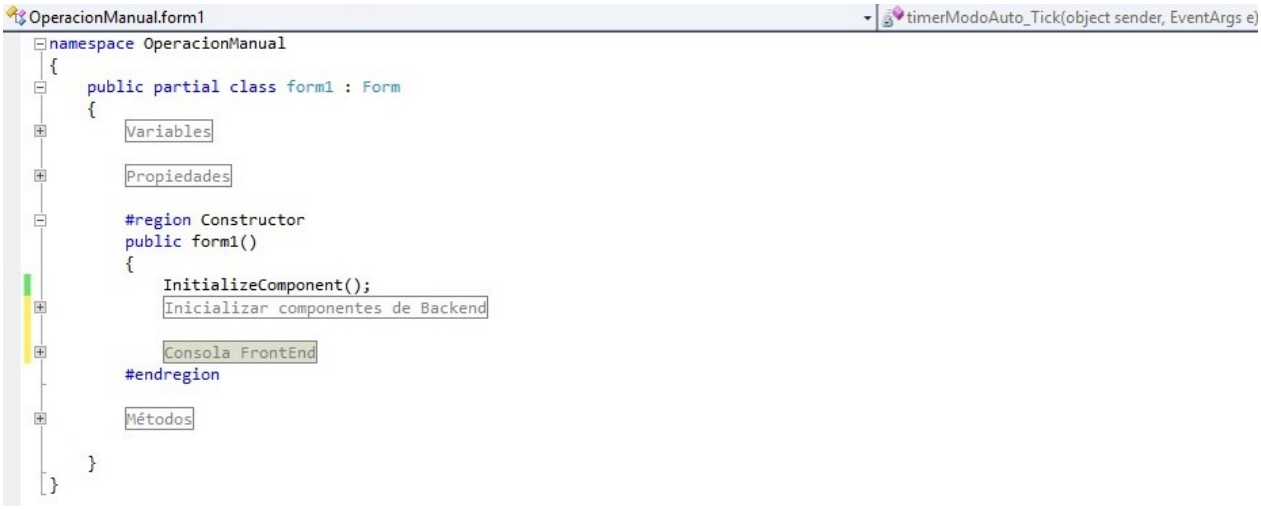


Figura 3.12: Código Estructura form1()- Constructor, Fuente: SCADA operando en Huatacondo

Como se puede apreciar en la Fig. 3.12 que contiene al código, la región del *"constructor"* tiene dos partes: *"inicializar componentes del backend"* y la *"consola del frontend"*.

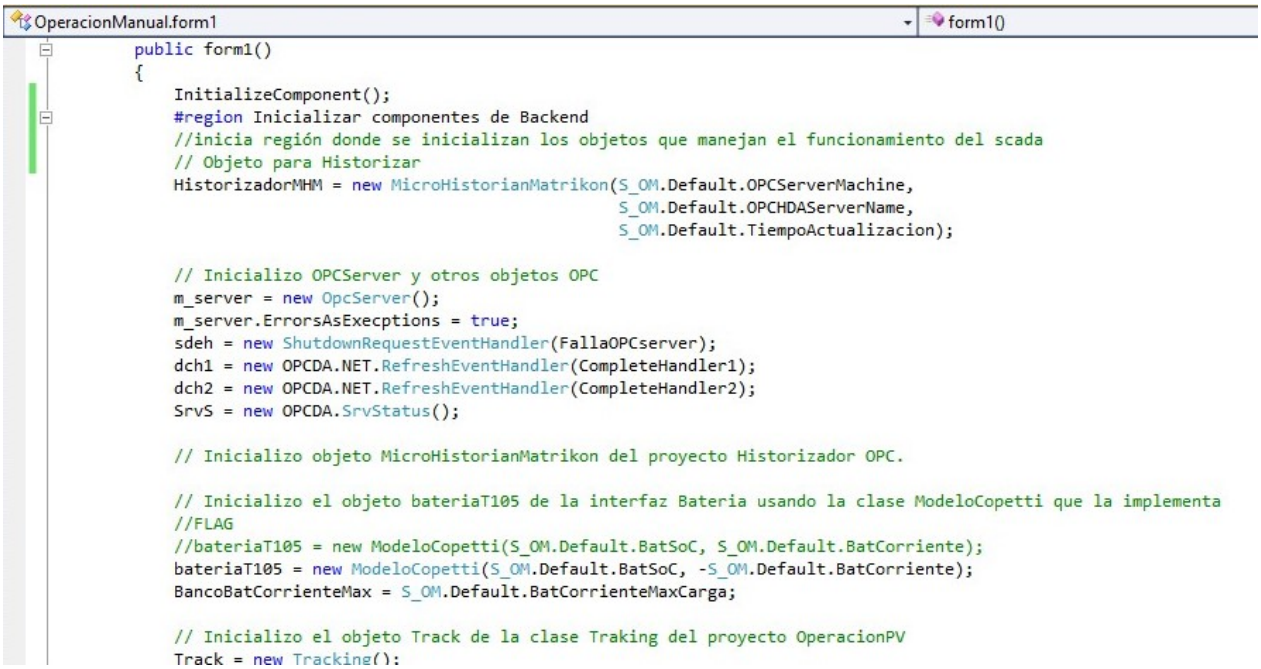


Figura 3.13: Código Estructura form1()- Constructor Backend(1), Fuente: SCADA operando en Huatacondo

Aquí se puede ver en las Fig. 3.13, 3.14 y 3.15 del código, que se inicializan componentes de las funciones del SCADA con las que a su vez inicializan el OPCServer, las baterías y el tracking que controla el sistema PV.

En las Fig. 3.13 y 3.14 se puede apreciar el desarrollo del **BackEnd**. En esta parte se presentan los siguientes procedimientos y/o funciones:

- Inicializar Historizador con la variable *HistorizadorMHM*, objeto MicroHistorianMatrikon
- Inicializar objetos OPCServer y otros objetos OPC
- Inicializar objetos relacionados con la clase de baterías
- Inicializar el objeto Track de la clase *Tracking* del proyecto Operación PV: Aquí se incluyen los botones de auto ON/OFF y también los botones manuales de ON/OFF
- Inicializar el SoC histórico: El SoC se historiza 1440 veces al mes, es decir, 2 veces por hora.

Luego de inicializado el *BackEnd*, se comienza con la *Consola FrontEnd* como se puede leer en las Fig. 3.15, 3.16, 3.17 y 3.18.

Aquí se presentan los siguientes procedimientos y/o funciones:

- Se inicializan los botones del ToolStrip (barra de herramientas)
- Se inicializan los botones de los "Modos de operación"
- Se inicializan los botones de y el *DateTimePicker* de la configuración automático/fijo: Esto tiene que ver con el funcionamiento del Diesel
- Se inicializan los botones del Data Logger

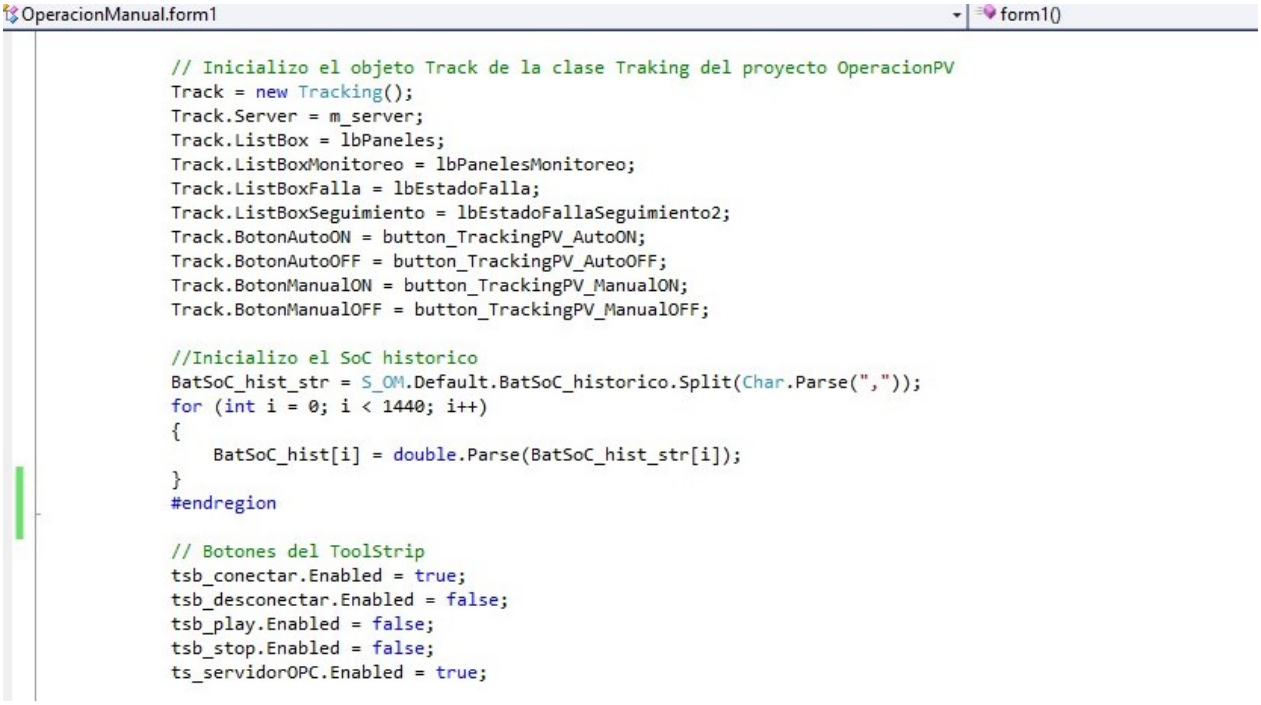
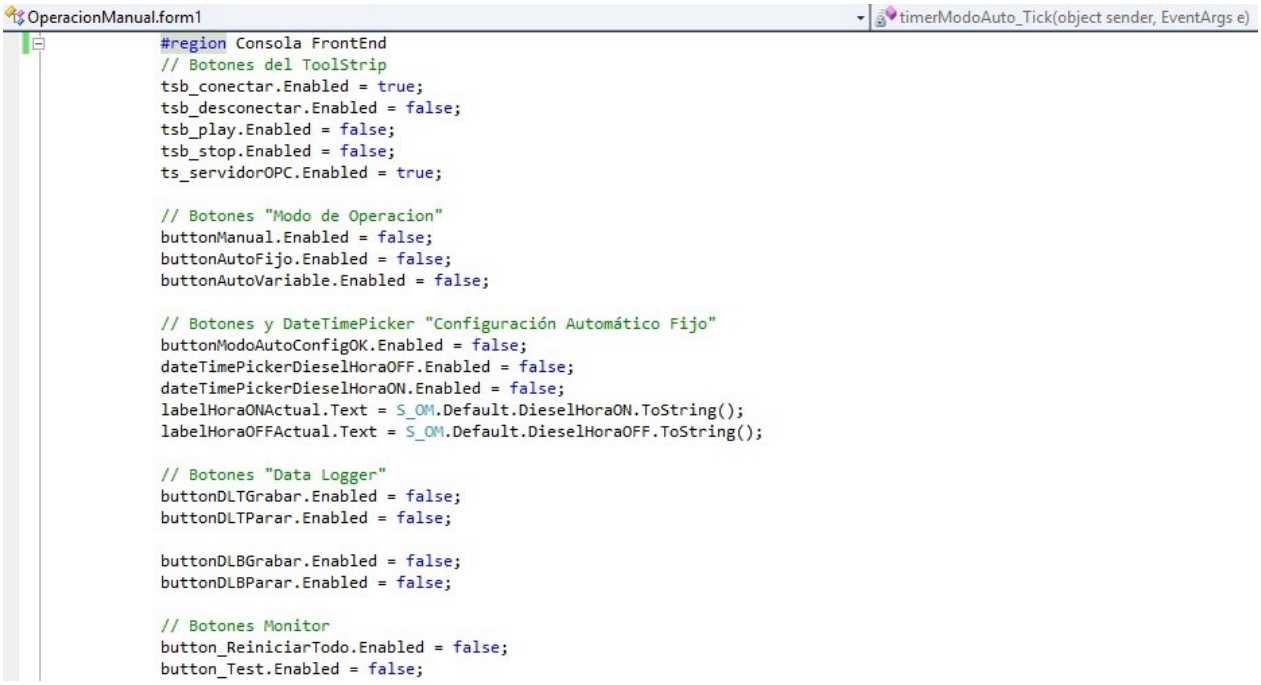


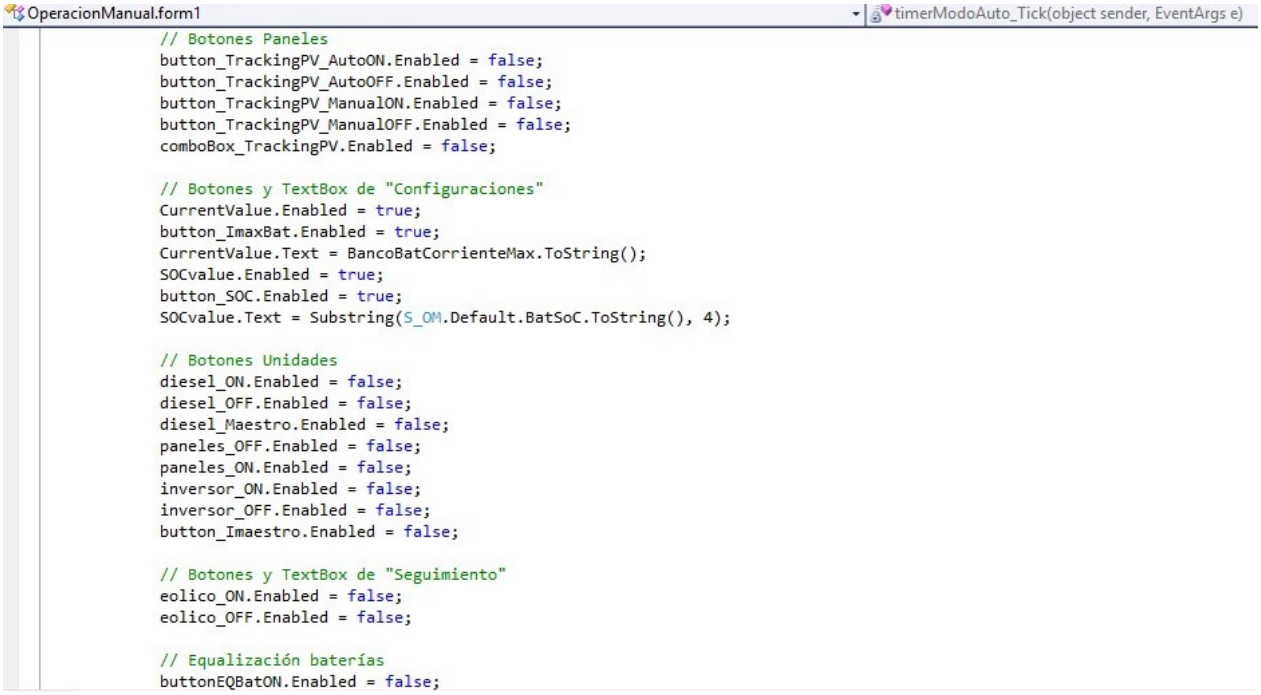
Figura 3.14: Código Estructura form1()- Constructor Backend(2), Fuente: SCADA operando en Huatacondo

- Se inicilizan los botones del Monitor
- Se inicializan los botones de los Paneles AutoON/AutoOFF y ManualON/ManualOFF
- Botones y "TextBox"de configuraciones: aquí están incluidas el SOC, CurrentValue y la Imax de la Batería.
- Botones para prender/apagar las unidades: diesel, paneles, inversor, eólico (que aparece en la sección de "Botones y TextBox de seguimientoz un botón maestro.
- Equalización de Baterías
- Inicialización de los modos de operacipon
- Inicialización de Labels de Estados y Valores de: Generadores Diesel, Planta PV (P1,P2 y P3), Eólico, Inversor (Estado y Potencia), Baterías (V,I y SoC)
- Labels Monitor: textos de componentes del sistema
- Componentes de la RED
- Función de datetimePicker para encender y apagar el Diesel
- Mensaje de bienvenida




```
#region Consola FrontEnd
// Botones del ToolStrip
tsb_conectar.Enabled = true;
tsb_desconectar.Enabled = false;
tsb_play.Enabled = false;
tsb_stop.Enabled = false;
ts_servidorOPC.Enabled = true;

// Botones "Modo de Operacion"
buttonManual.Enabled = false;
buttonAutoFijo.Enabled = false;
buttonAutoVariable.Enabled = false;

// Botones y DateTimePicker "Configuración Automático Fijo"
buttonModoAutoConfigOK.Enabled = false;
dateTimePickerDieselHoraOFF.Enabled = false;
dateTimePickerDieselHoraON.Enabled = false;
labelHoraONActual.Text = S_OM.Default.DieselHoraON.ToString();
labelHoraOFFActual.Text = S_OM.Default.DieselHoraOFF.ToString();

// Botones "Data Logger"
buttonDLTGrabar.Enabled = false;
buttonDLTParar.Enabled = false;

buttonDLBGrabar.Enabled = false;
buttonDLBParar.Enabled = false;

// Botones Monitor
button_ReiniciarTodo.Enabled = false;
button_Test.Enabled = false;
```

Figura 3.15: Código Estructura form1()- Constructor Consola FrontEnd (1), Fuente: SCADA operando en Huatacondo




```
// Botones Paneles
button_TrackingPV_AutoON.Enabled = false;
button_TrackingPV_AutoOFF.Enabled = false;
button_TrackingPV_ManualON.Enabled = false;
button_TrackingPV_ManualOFF.Enabled = false;
comboBox_TrackingPV.Enabled = false;

// Botones y TextBox de "Configuraciones"
CurrentValue.Enabled = true;
button_ImaxBat.Enabled = true;
CurrentValue.Text = BancoBatCorrienteMax.ToString();
SOCvalue.Enabled = true;
button_SOC.Enabled = true;
SOCvalue.Text = Substring(S_OM.Default.BatSoC.ToString(), 4);

// Botones Unidades
diesel_ON.Enabled = false;
diesel_OFF.Enabled = false;
diesel_Maestro.Enabled = false;
paneles_OFF.Enabled = false;
paneles_ON.Enabled = false;
inversor_ON.Enabled = false;
inversor_OFF.Enabled = false;
button_Imaestro.Enabled = false;

// Botones y TextBox de "Seguimiento"
eolico_ON.Enabled = false;
eolico_OFF.Enabled = false;

// Equalización baterías
buttonEQBatON.Enabled = false;
```

Figura 3.16: Código Estructura form1()- Constructor FrontEnd(2), Fuente: SCADA operando en Huatacondo

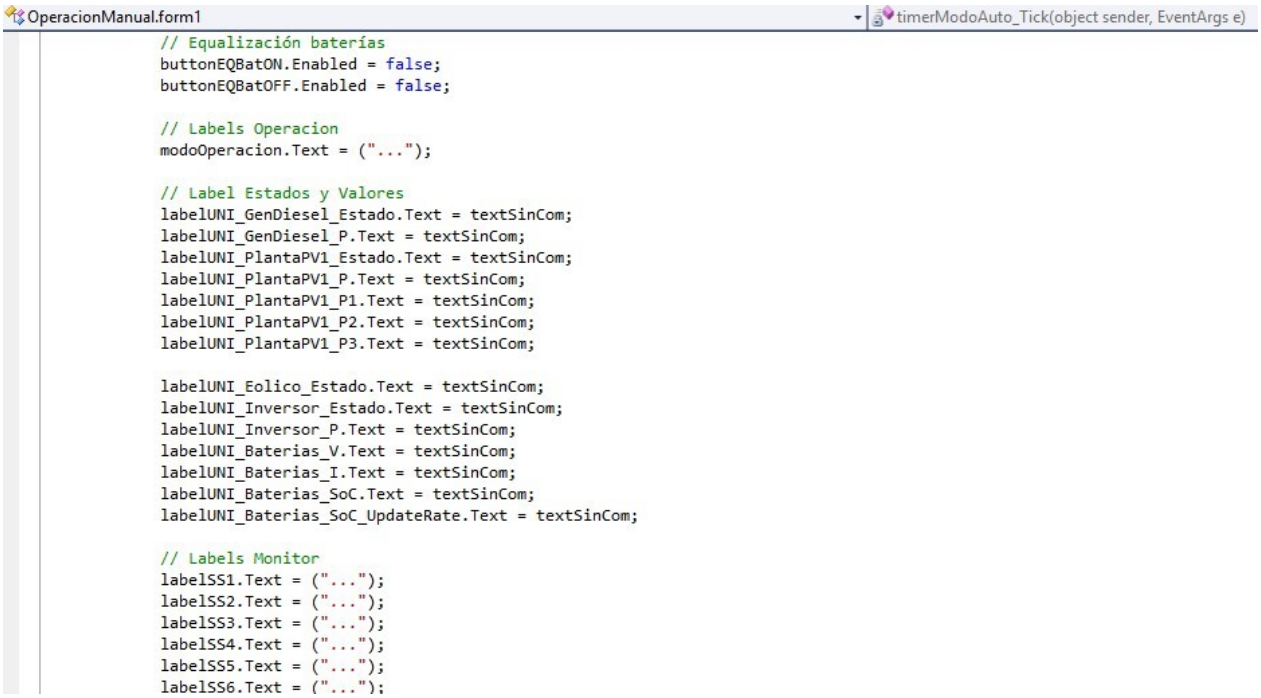


Figura 3.17: Código Estructura form1()- Constructor FrontEnd(3), Fuente: SCADA operando en Huatacondo

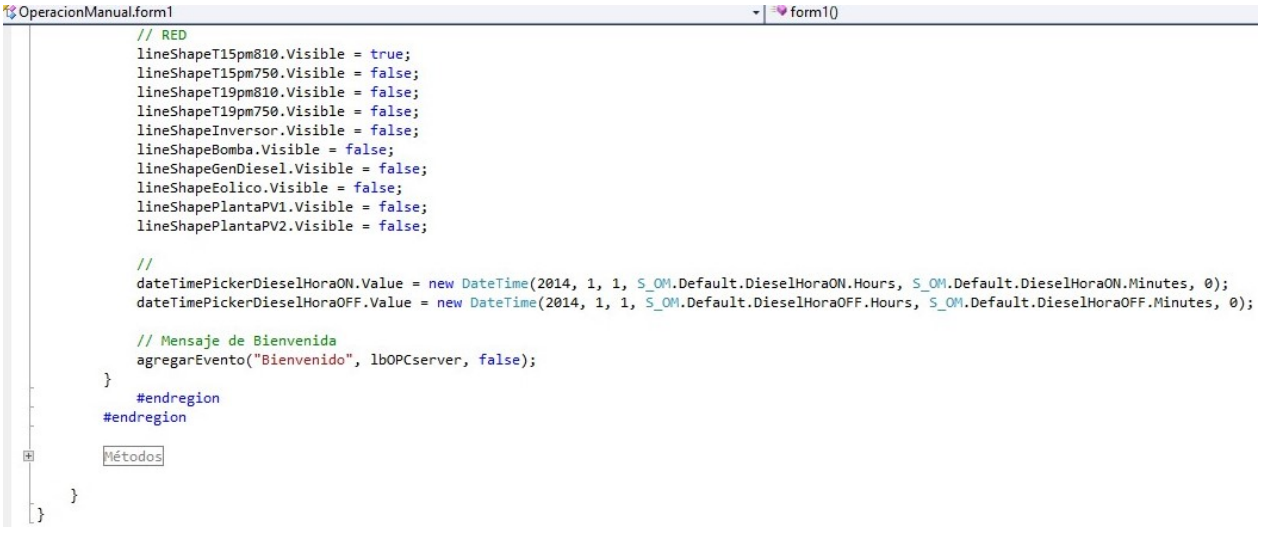


Figura 3.18: Código Estructura form1()- Constructor FrontEnd(4), Fuente: SCADA operando en Huatacondo

Métodos

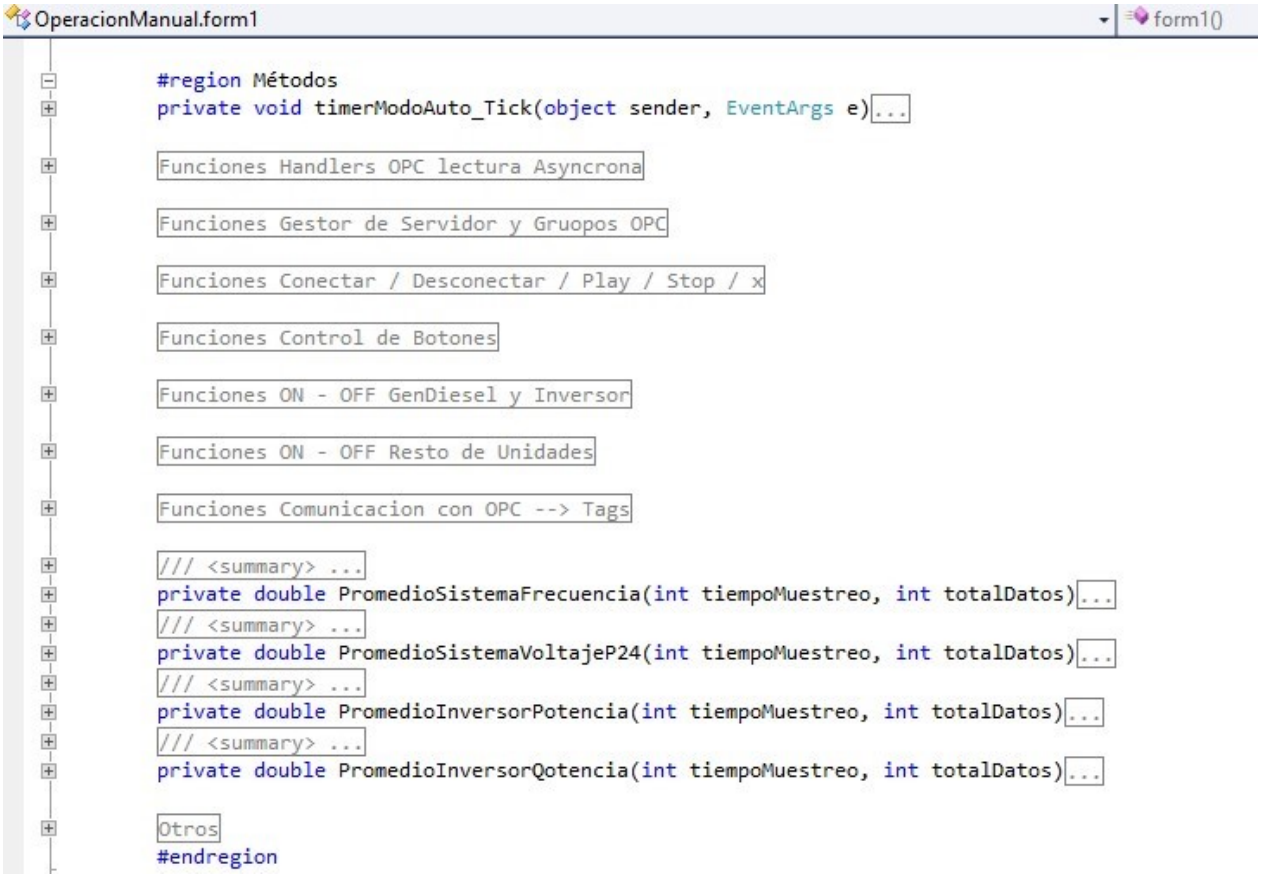


Figura 3.19: Código Estructura form1()- Métodos - Vista general de Funciones, Fuente: SCADA operando en Huatacondo

Aquí se aplican las condiciones de *if* para saber si el tiempo que guarda la variable "Tauto" es efectivamente igual al tiempo donde ocurre el proceso, eso indicará si se prende o apaga el diésel.

```
(diesel_ON.Enabled=false;diesel_OFF.Enabled=false;)
```

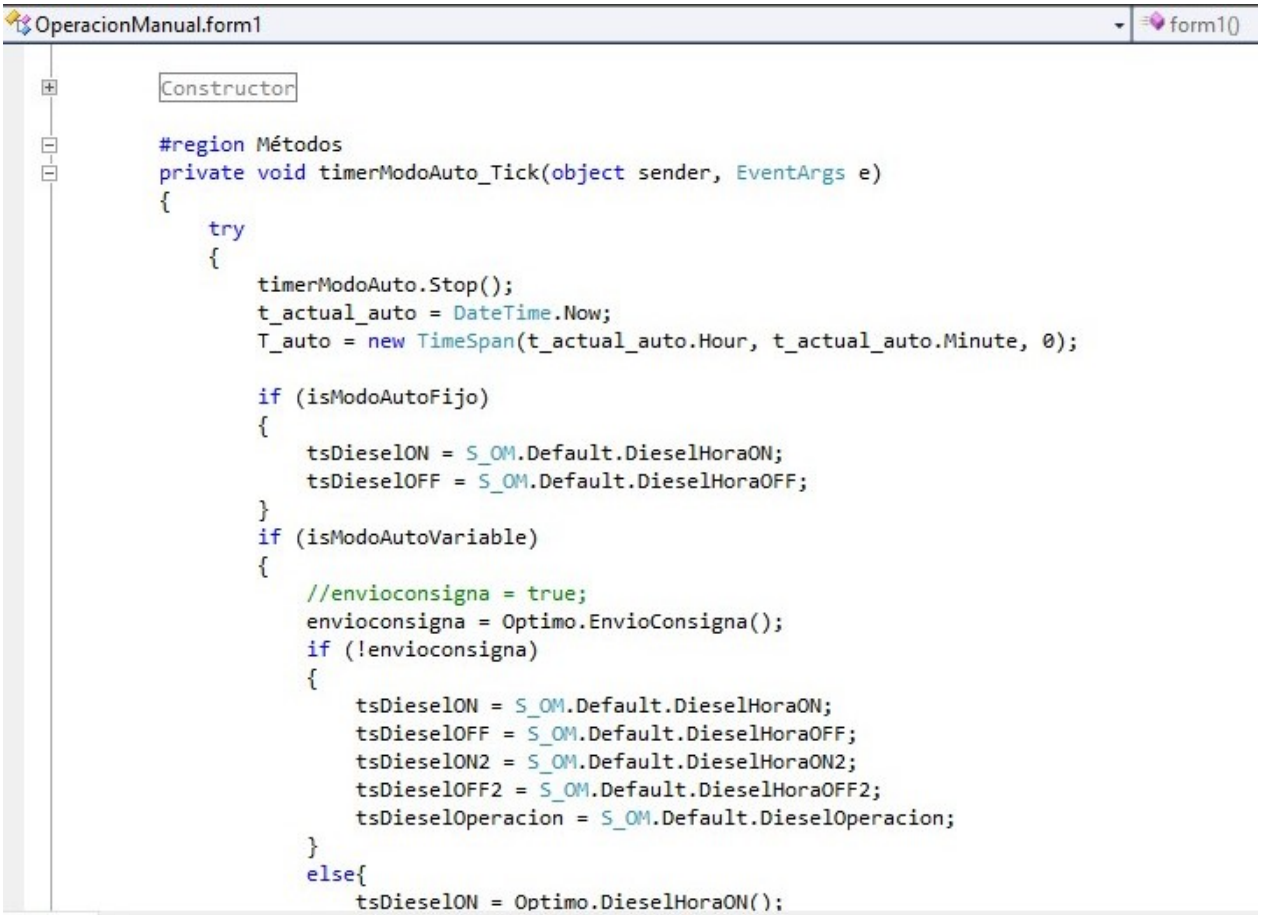


Figura 3.20: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

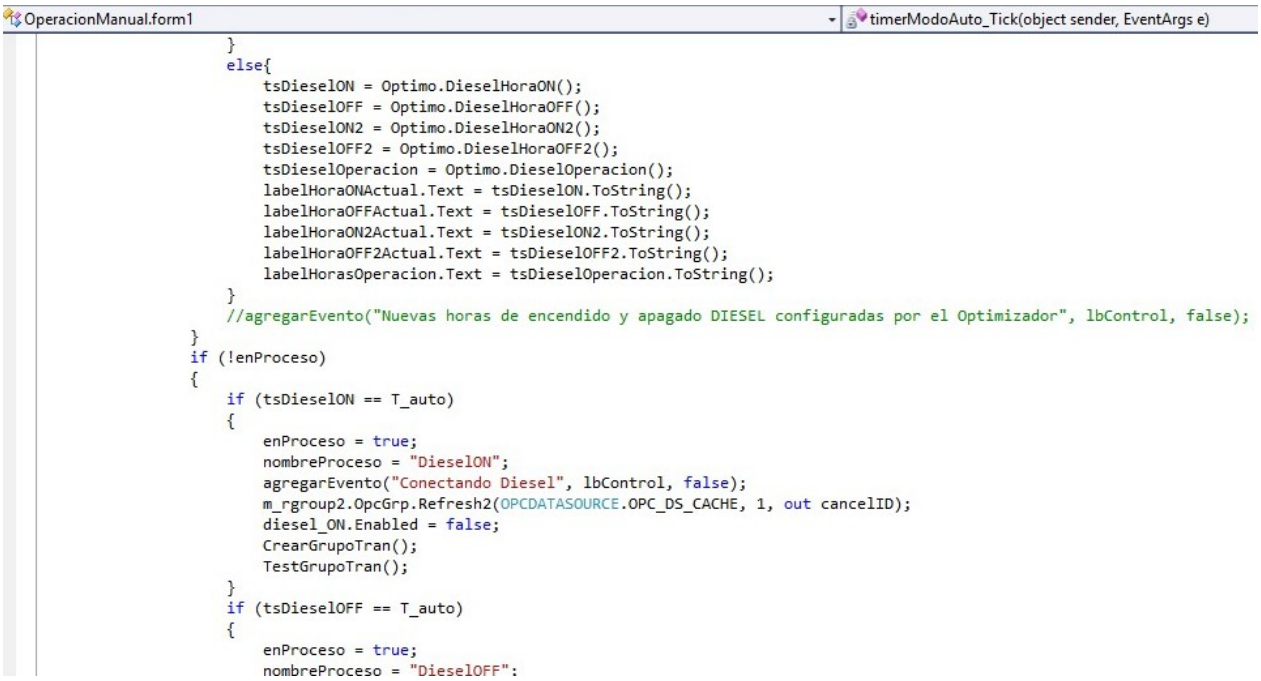


Figura 3.21: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

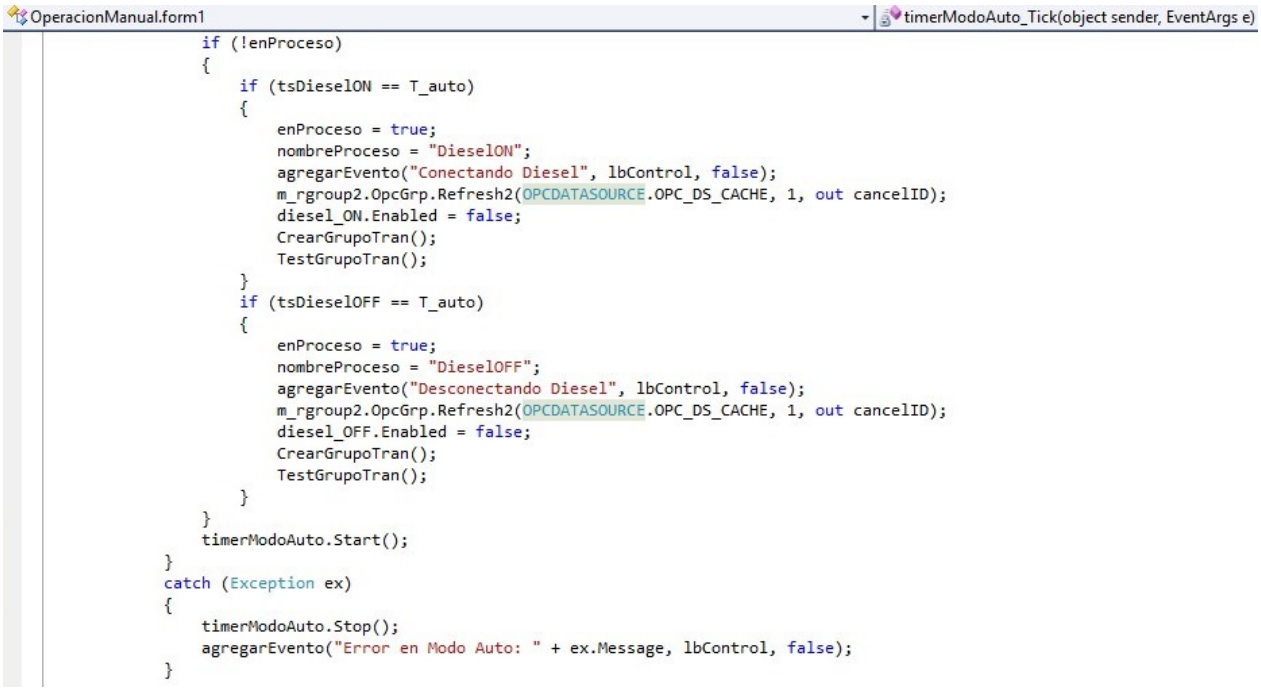


Figura 3.22: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

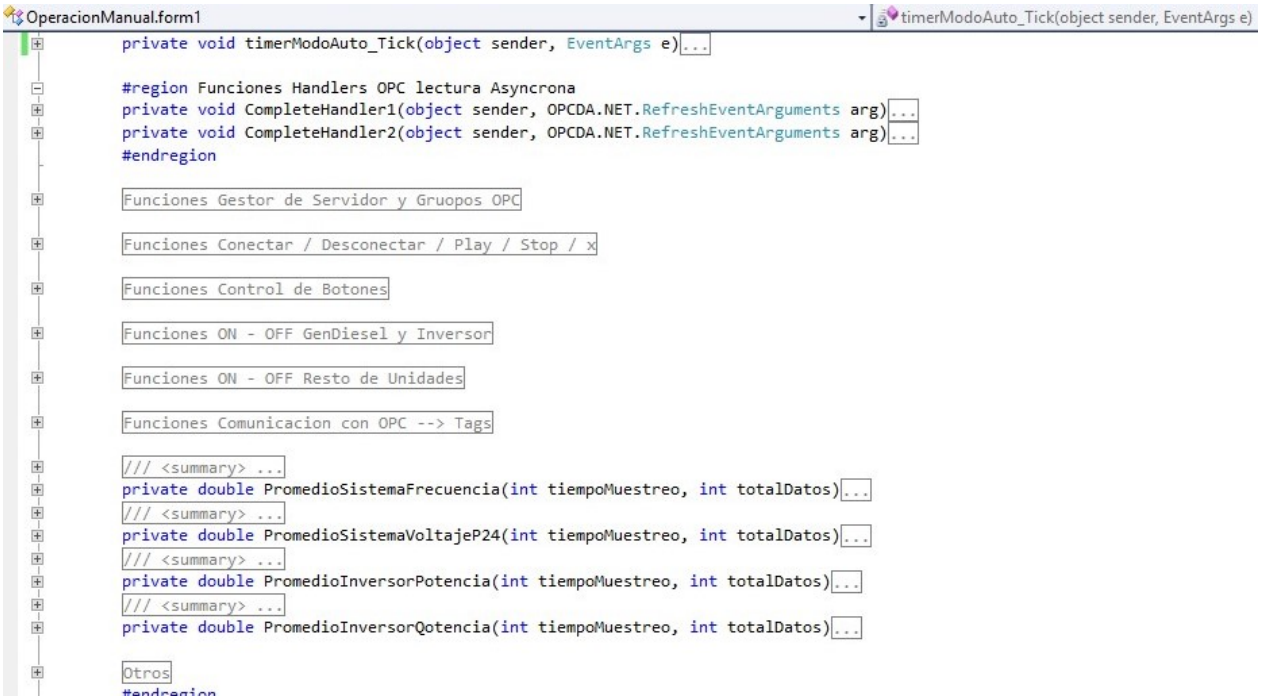


Figura 3.23: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

Se puede ver que en la Fig. 3.23 se declaran dos

‘‘`private void CompleteHandlerX(object sender, OPCdA.neT.ResfreshEventsArguments arg)..`

, donde X corresponde al 1 y 2.

En los cuales se listan las siguientes funciones:

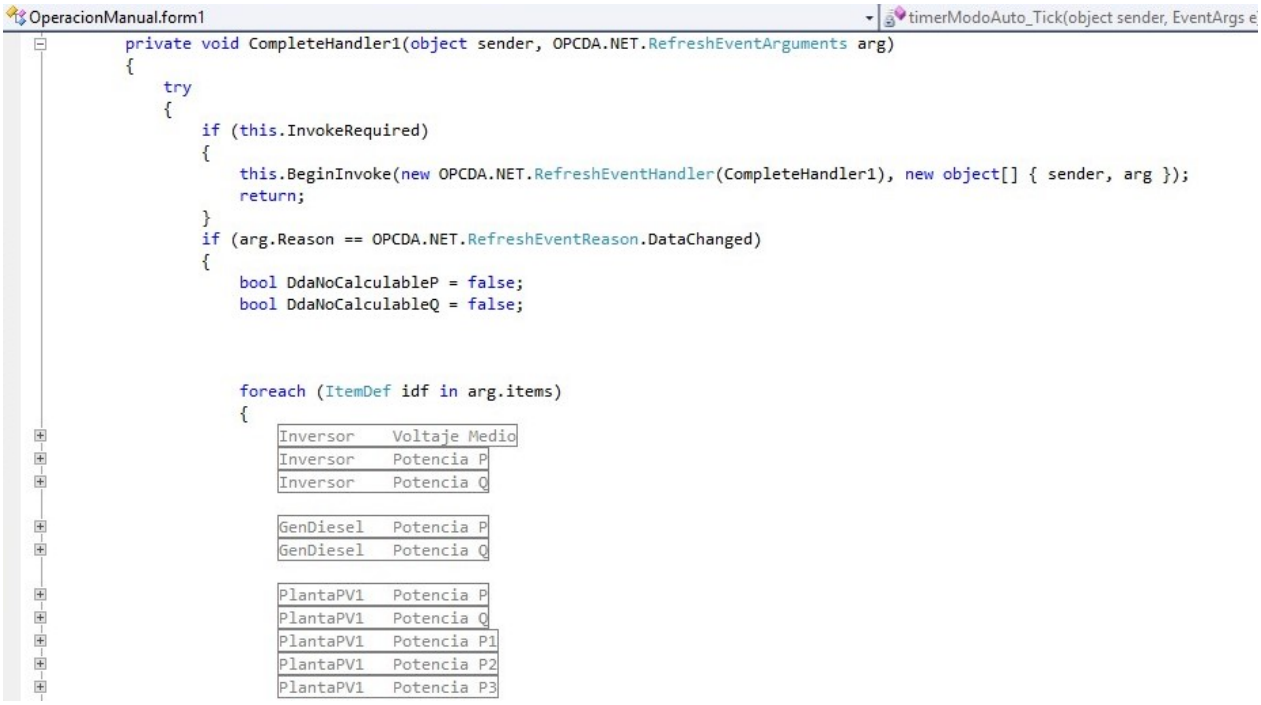


Figura 3.24: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

El método InvokedRequired se encarga de confirmar si la variable que controla la funcion true if the control's Handle was created on a different thread than the calling thread (indicating that you must make calls to the control through an invoke method); otherwise, false.

Luego se definen las *regions* que reciben los parámetros de los inversores, los generadors diesel, las plantas(arreglos) PV (3) y la frecuencia, voltaje y potencia del sistema completo de Huatacondo.

El archivo está dividido en secciones. Una idea quizás sería agrupar los ítems por “sistema”, es decir, pv, diesel, baterías, etc. O bien, por parámetros: voltaje, corriente, potencia, frecuencia.

En la región de “inversor Estados” se puede ver que se ponen las diferentes opciones para cuando el inversor está encendido o apagado. Cada región contiene mas menos las mismas opciones, las de estados de botones y también el mostrar/desplegar los mensajes de “Encendido” o “Apagado”.

Se puede ver en esta imagen que al igual ue los m+etodos implementados para el inversor,

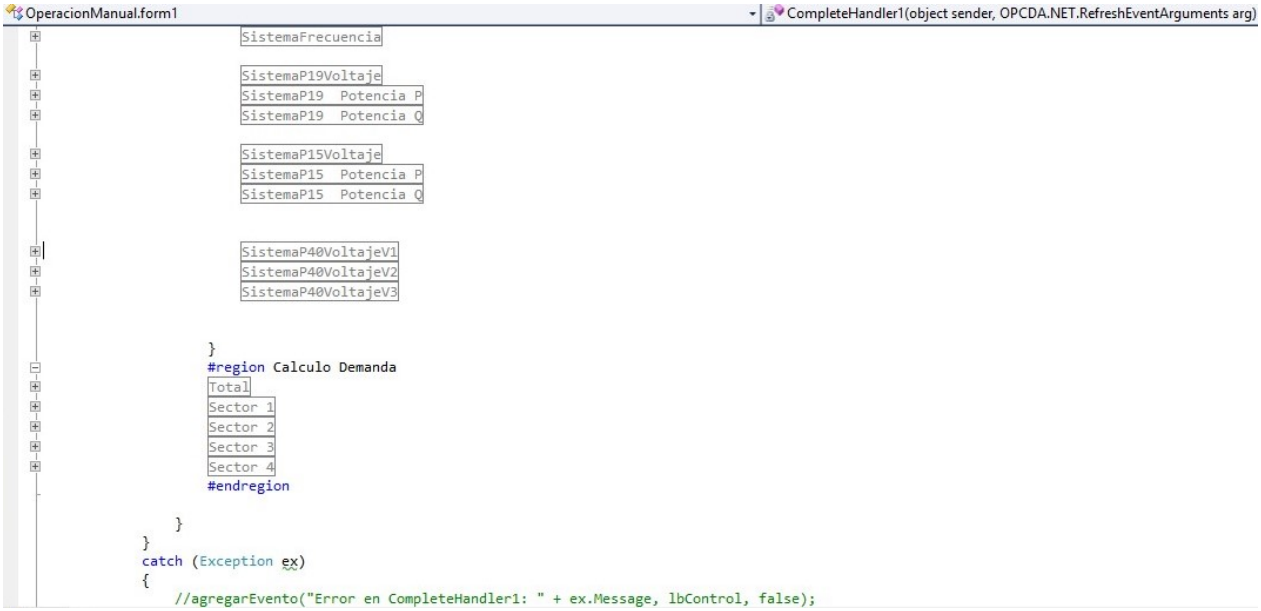


Figura 3.25: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

la clase de "Planta PV" también tiene estads de "conectado" y "Desconectado", además de tener una opción donde se escribe que hace cada botón.

Aquíse puede apreciarse una comparación de los estados que muestran si la planta PV1 y el Eólico se encuentra apagada o encendida. Se puede dar cuenta que los métodos son similares y su estructura de código es la misma.

En esta parte aparecen las funciones que se encargan de conectar/desconectar el servidor , crear/cerrar grupos de monitoreo y otras funciones.

En esta parte están concentradas las Funciones del Gestor de Servidor y grupos OPC. Esta clase desplegada se utiliza para conectar el servidor y en los métodos dentro de esta se detallan las instrucciones del qúe hacer cuando se ocupa el "ConectarServidor()"

Aquí en la segunda imagen de la parte que maneja el OPC se puede ver que el programa arroja mensajes de desconexión del servidor y contiene excepciones en caso de que falle el proceso, con un mensaje de error "no es posible desconectar. . . "

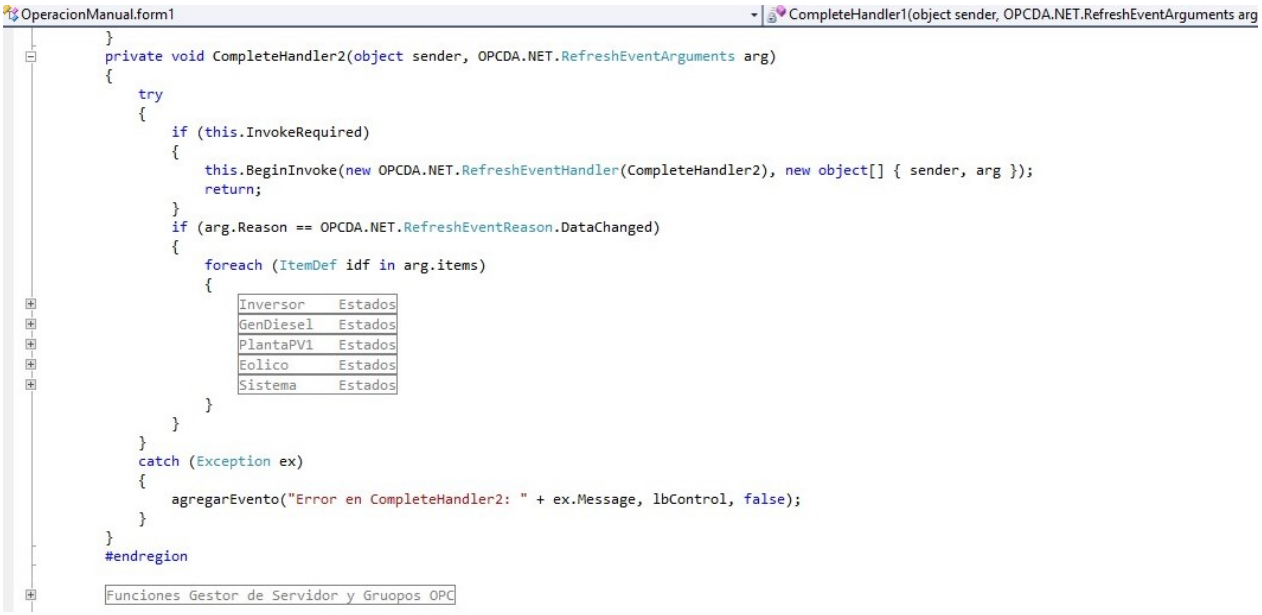


Figura 3.26: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

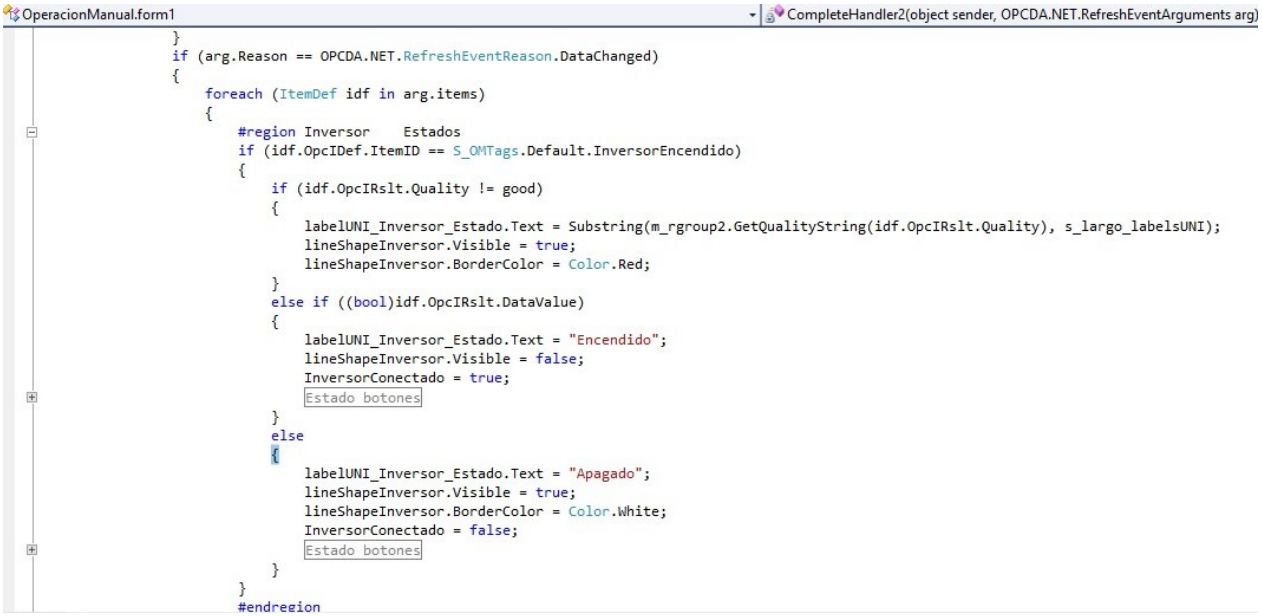


Figura 3.27: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

Figura 3.28: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

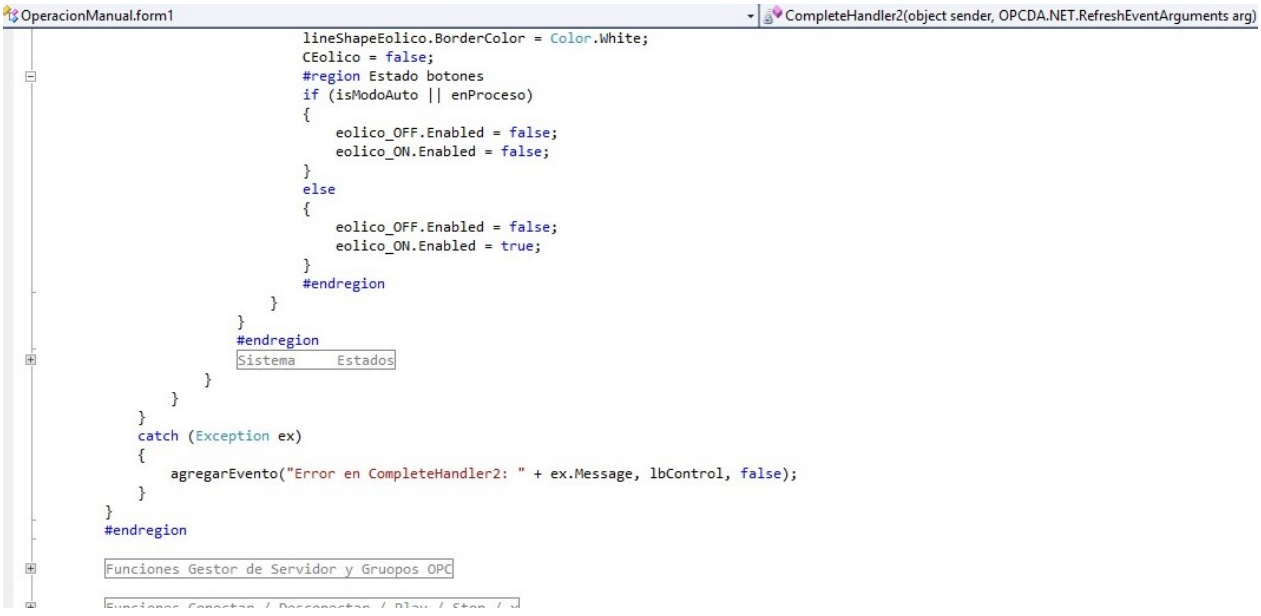


Figura 3.29: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

Figura 3.30: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

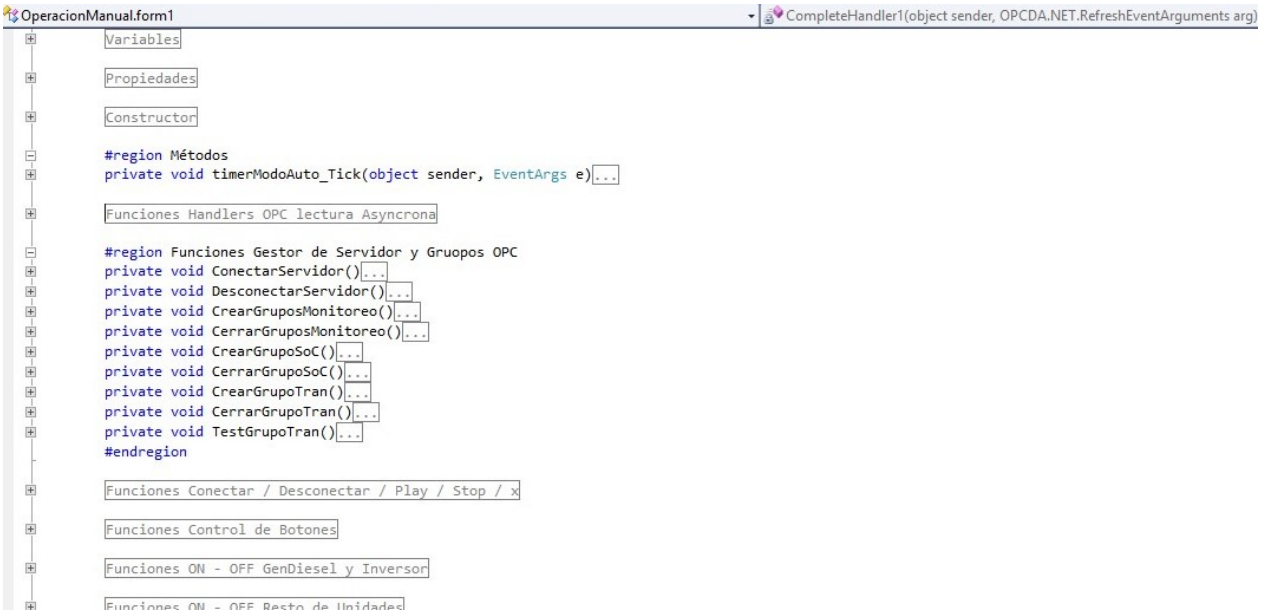


Figura 3.31: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

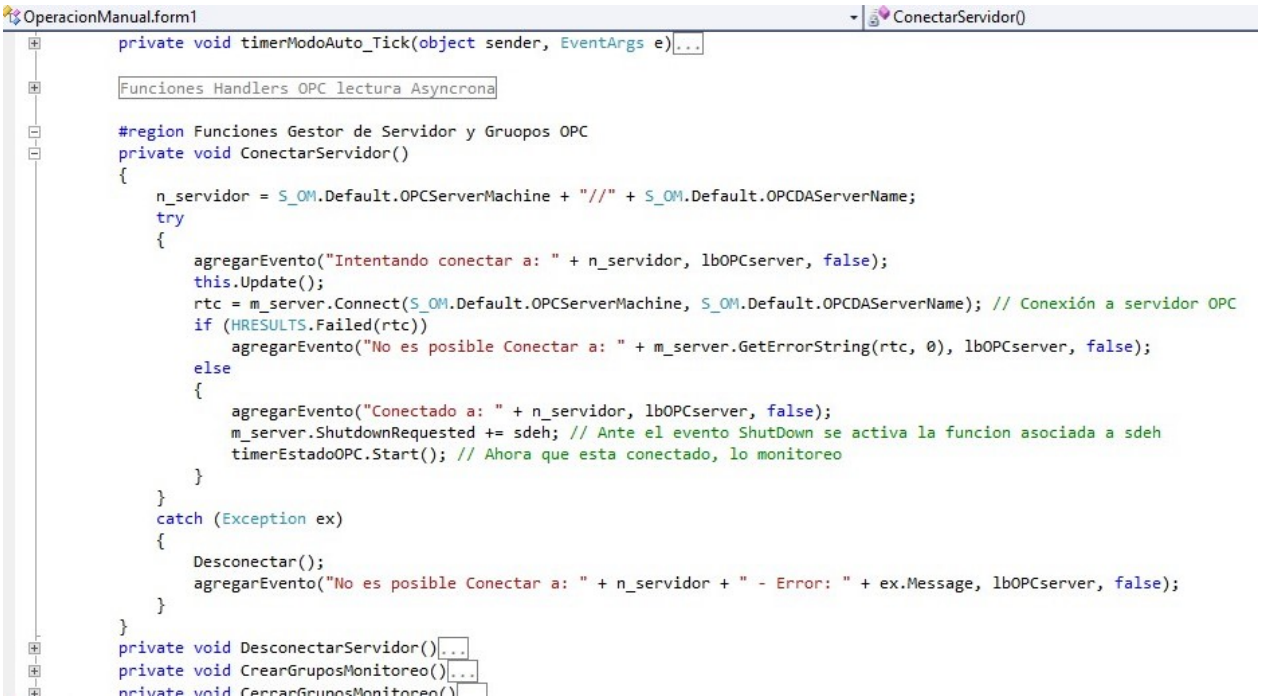


Figura 3.32: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

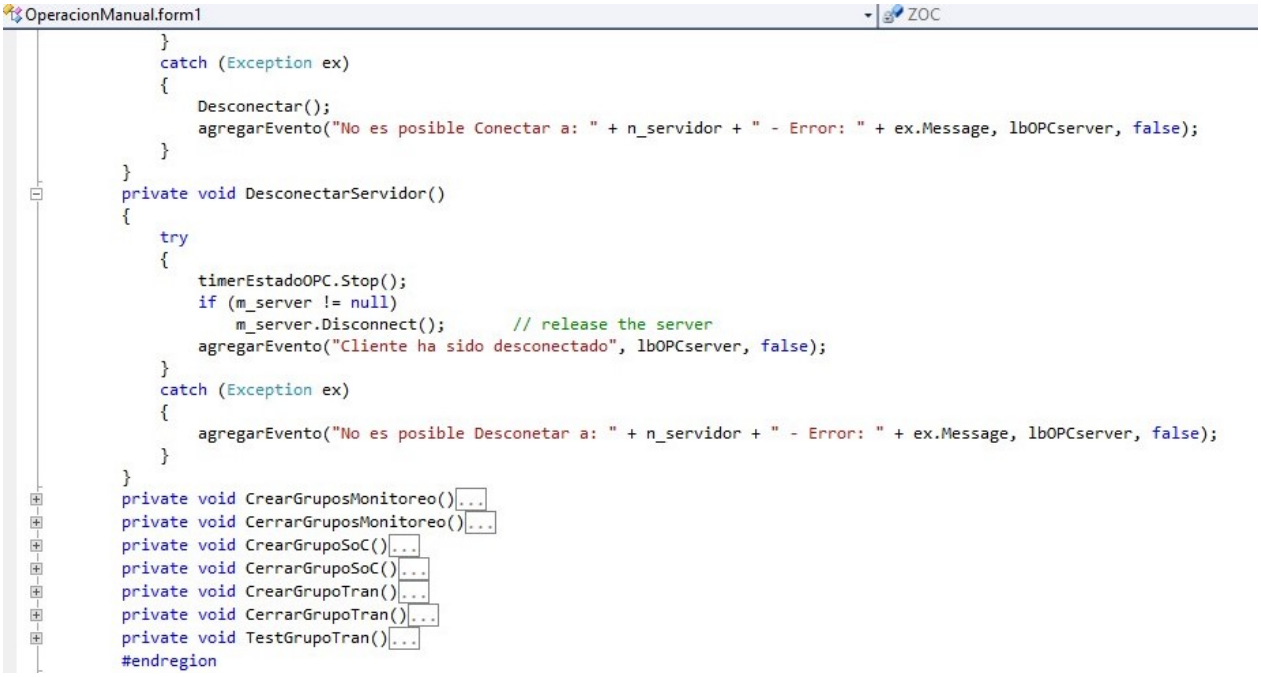


Figura 3.33: Código Estructura form1()- Métodos - Funciones Handlers OPC Lectura asincrona, Fuente: SCADA operando en Huatacondo

3.2. Identificación de Problemas y Oportunidades

Como se aprecia en las Figuras del código de la sección anterior, se sigue poniendo todas las variables del problemas mezcladas: tanto los valores que captura y muestra el programa en el *FrontEnd* como la diagramación de las ventanas y las funciones que operan tras el programa, es decir, el *BackEnd*.

Al mismo tiempo hay variables que son llamadas desde dentro de la sección del programa *"form1"* que al modularizar el programa se debe tener cuidado de tener en consideración desde donde y hacia donde llaman objetos/funciones.

3.2.1. Desglose de funciones que Operan en el archivo *"form1"* SCADA

Según las principales secciones del código del *"form1"* este se divide en 4 secciones principales:

- *Variables:* En esta sección se pueden encontrar la declaración de variables:
  - Asociadas al funcionamiento general del SCADA
  - Asociadas al DateTime y TimeSpam
  - Variables del Diesel, Inversor y la Red.
  - Baterías
  - OPC y Servidor
  - Datalogger
  - Calculo de SOC
  - Historizador
- *Propiedades:* En esta sección principalmente hay funciones *ListBox* asociadas a los paneles soalres, estados de falla y seguimiento a los estados de falla (4 ListBox).
- *Constructor:* Principalmente en esta sección se encuentran entremezclados partes del Backend y del FrontEnd. Hay presentes elementos asociados a los paneles solares y también a contadores referidos al proceso de control del BESS.
- *Métodos:* Los métodos en general están orientados en esta sección a:
  - Gestor de servidor y OPC
  - Funciones de conexión, desconexión, stop y play
  - Funciones de control de botones (FrontEnd)
  - Funciones de ON/OFF de las unidades (2 bloques)
  - Funciones de comunicación con OPC y Tags.

3.2.2. Clasificación de funciones segun el rol que desempeñan

Las funciones se dividen principalmente en funciones que ejecutan procesos de Backend como son las operatorias y en funciones que modifican el Frontend, particularmente los botones.

Además existen divisiones dentro del mismo archivo que refieren a las *regiones*, donde viven funciones y variables asociadas a los módulos de los que se habla más arriba: Inversor, Paneles, Historizador, Baterías, OPC y el Diesel.

3.2.3. Identificación de Problemas y Oportunidades

Dentro de los problemas que se identifican en el código, es que están mezcladas las funciones que deberían operar en el Backend con las que deben operar en el Frontend. Este problema se produjo en diseños anteriores, ya que este programa ha sido desarrollado por diferentes estudiantes memoristas que lo han tomado, hecho modificaciones y anexado nuevos archivos. Sin embargo, hay secciones en el código donde se agrupan funciones relacionadas a partes completas que deben ir al Backend o Frontend. Por ejemplo, la región *Gestor de Servidor y OPC*, contiene:

- private void ConectarServidor()
- private void DesconectarServidor()
- private void CrearGruposMonitoreo()
- private void CerrarGruposMonitoreo()
- private void CrearGrupoSoC()
- private void CerrarGrupoSoC()
- private void CrearGrupoTran()
- private void CerrarGrupoTran()
- private void TestGrupoTran()

Esta funciones se encargan de la gestión del servidor y del OPC, y cumplen funciones de monitoreo, control y conexión con el servidor. A continuación se muestra el ejemplo de la función *ConectarServidor()* en (3.1)

Listing 3.1: Código de función *ConectarServidor()* C♯

```
private void ConectarServidor()
        {
            n_servidor = S_OM.Default.OPCServerMachine + "//" +
               S_OM.Default.OPCDAServerName;
            try
            {
                agregarEvento("Intentando conectar a: " +
                   n_servidor, lbOPCserver, false);
                this.Update();
                rtc = m_server.Connect(S_OM.Default.
                   OPCServerMachine, S_OM.Default.OPCDAServerName
                   ); // Conexi n a servidor OPC
                if (HRESULTS.Failed(rtc))
                    agregarEvento("No es posible Conectar a: " +
                        m_server.GetErrorString(rtc, 0),
                       lbOPCserver, false);
                else
                {
                    agregarEvento("Conectado a: " + n_servidor,
                       lbOPCserver, false);
                    m_server.ShutdownRequested += sdeh; // Ante
                       el evento ShutDown se activa la funcion
                       asociada a sdeh
                    timerEstadoOPC.Start(); // Ahora que esta
                       conectado, lo monitoreo
                }
            }
            catch (Exception ex)
            {
                Desconectar();
                agregarEvento("No es posible Conectar a: " +
                   n_servidor + " - Error: " + ex.Message,
                   lbOPCserver, false);
            }
        }
```

3.3. Propuesta de Mejoras

A continuación se tratará respecto a las propuestas de modularización y mejoras al código del SCADA. Las modificaciones que aparecen a continuación tienen lugar en el documento *form1* que se muestra en la Fig. 3.34

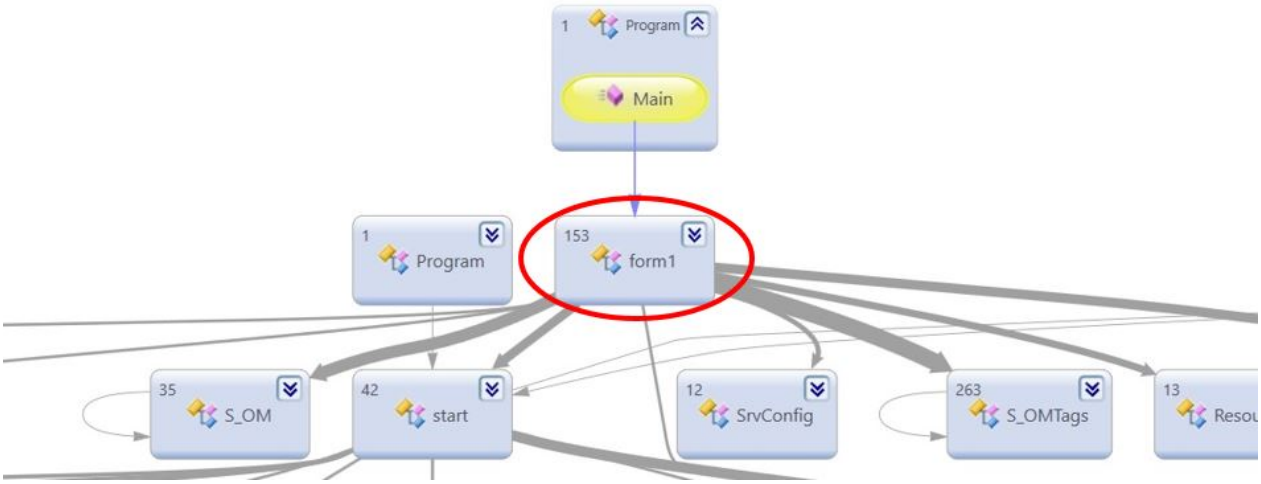


Figura 3.34: Diagrama de Clases SCADA Huatacondo

3.3.1. Propuesta de módulos concretos

En particular y dado que los problemas principales de "desorden de código" se detectaron en el documento llamado *form1*, es en donde se propone generar los siguientes módulos:

- BackEnd: Contiene todos los elementos que funcionan tras las operaciones que se hacen en el SCADA
- FrontEnd: Contiene todos los elementos que operan en la ventana del operador
- Módulos Varios: Contiene todo el resto de elementos que no se encuentra en las dos categorías anteriores y que sí se categorizan dentro de los modulos propuestos anteriormente:
  - Diesel
  - PV
  - Baterías
  - Inversor
  - Eólico
  - Historizador

Se debe notar que para los alcances de esta memoria solo se realizará la modularización del código separándolo en Frontend y Backend.

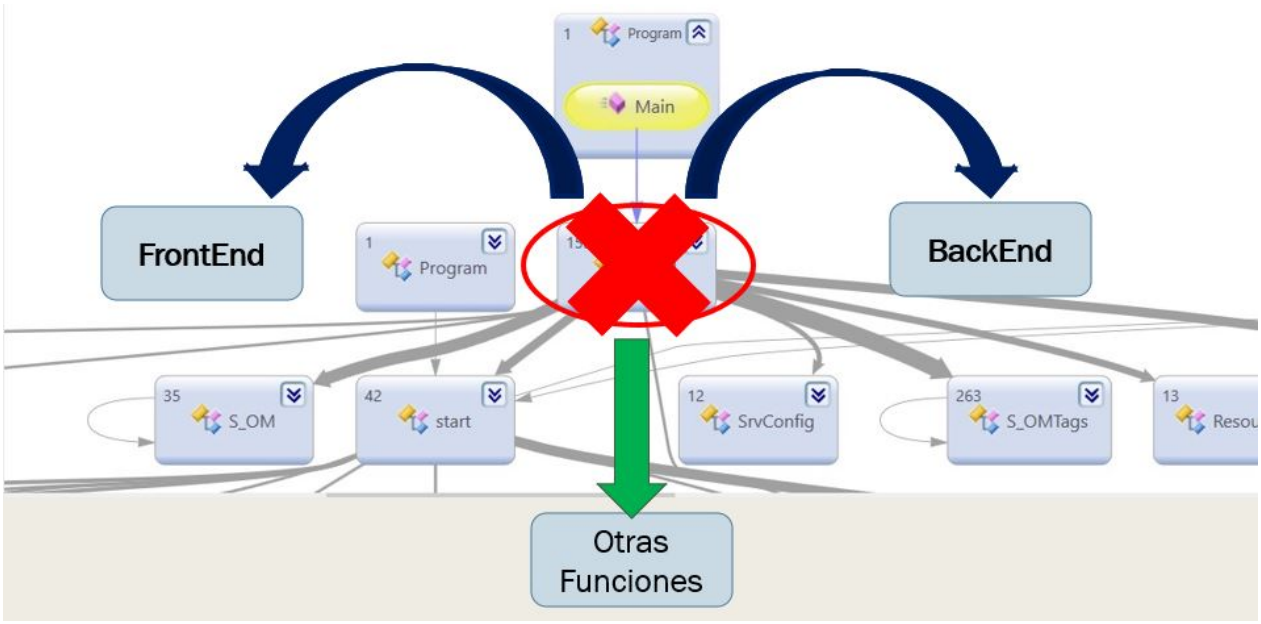


Figura 3.35: Diagrama de separación del form1 en Backend y FrontEnd

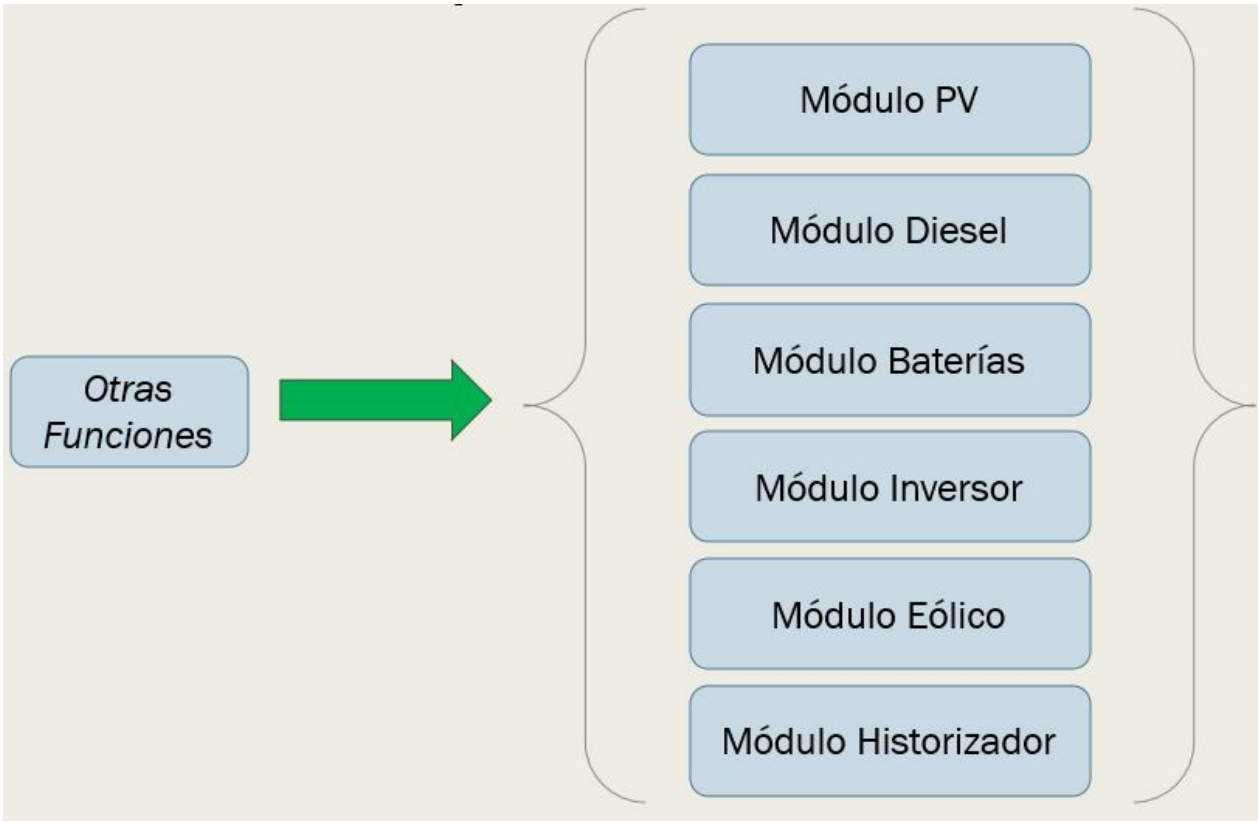


Figura 3.36: Diagrama de módulos que se desprenden del form1

De esta manera la distribución de las funciones que actualmente operan en el *form1* quedaría así:

FrontEnd

Las funciones que componen al FrontEnd serían las siguientes:

- Variables:
  - –
- Propiedades
  - *ListBox* asociadas a los paneles solares, estados de falla y seguimiento a los estados de falla (4 ListBox).
- Constructor
  - –
- Métodos
  - Funciones de control de botones (FrontEnd)
  - Funciones de ON/OFF de las unidades (2 bloques)

BackEnd

Las funciones que componen al BackEnd queda de la siguiente manera:

- Variables:
  - Constante y Variables de Red
  - OPC Refresh Group
  - Estado de los contactores
  - Potencia de los PMs
  - Variables para cálculo de SOC
  - OPC SyncGroup (cálculo del SoC)
  - Generales
- Propiedades
- Constructor
  - Inicialización variable *m_server*
  - Inicialización variable *deh*
- Métodos
  - Gestor de servidor y OPC
  - Funciones de conexión, desconexión, stop y play
  - Funciones de comunicación con OPC y Tags.

Módulos concretos

Las funciones que corresponden a los módulos queda clasificada de la siguiente manera:

- Baterías
  - Cálculo de SOC (Variables)
- Diesel
- Fotovoltaico (PV)
- Inversor
- Eólico (Este módulo se encuentra deshabilitado en esta versión del programa)
- Historizador

3.3.2. Nuevo código

Para implementar el nuevo código, como se propuso anteriormente, se realizará una copia del archivo *"form1"* para modificarla y que sea el nuevo FrontEnd propuesto. Para ello se seguirá la siguiente metodología:

- Se tomará una línea de código que se quiera agregar al Backend y se separará del resto del *"form1"*
- Luego se copiarán todas las funciones/definiciones relacionadas al Backend
- Se borrarán/comentarán variables del *"form1"* relacionadas al FrontEnd
- Se probará el programa. En caso de falla se deberán borrar la variable/función y agregarla al Backend o bien se modificará la variable y se llamará desde el Backend como *"Backend.variable"*. Estas serán denominadas variables globales
- Iterar. Si funciona volver al inicio hasta completar la modularización

3.3.3. Nuevas estructuras

Se propone que en un trabajo futuro existan nuevas estructuras como módulos pertenecientes a módulos FV, eólico, baterías, etc. Sin embargo, en esta primera iteración, solo se creará un nuevo archivo que tendrá como función ser el BackEnd, dejando al archivo *"form1"* como FrontEnd.

Para comenzar la modularización, se creó una instancia de clase llamada Backend (3.2)

Este proceso es iterativo. Se deben tomar variables del FrontEnd e ir agregándolas al Backend. Cada vez que se agrega una variable del Frontend al Backend, debe declararse e inicializarse en el Backend y borrarse su declaración e inicialización en el FrontEnd; por otro lado, las variables que no se puedan pasar al Backend, se llamarán en el Backend con la estructura *"frontEnd.variable"*, donde *"frontEnd"* es un objeto *"form1"*.

Listing 3.2: Código de clase Backend C♯

```
public class Backend
    {
        public form1 frontEnd;

        #region Constructor
        public Backend(form1 form)
        {
            frontEnd = form;
        }

    }
```

Además se crea el objeto miBackend de la clase Backend en el FrontEnd (3.3), el cual sirve de nexo para poder llamar a las variables del FrontEnd desde el Backend.

Listing 3.3: Código inicializar variable miBackend de clase Backend C♯

```
#region Constructor //Constructor del FrontEnd
        public form1()
        {
            InitializeComponent();
            #region Inicializar componentes de Backend

            miBackend = new Backend(this);
        }
```

El paso siguiente de la modularización fue mover los objetos y funciones relacionadas al servidor OPC, es decir la comunicación del programa SCADA, desde el FrontEnd al Backend. Para esto se comenzó moviendo las variables disponibles en el *Constructor* del FrontEnd de manera tal que pudiese seguir funcionando la comunicación del programa con el servidor OPC como se muestra en (3.4).

En este caso en particular se comenzó moviendo la variable *"m_server"*, la cual ahora se llama desde el FrontEnd como *"miBackend.m_server"*. Esta variable es muy importante ya que es responsable de la comunicación con el servidor.

Listing 3.4: Código cambio de variable desde FrontEnd al Backend C♯

```
public class Backend
{
    public form1 frontEnd;

    #region Variables
    public OpcServer m_server;
    #endregion

    #region Constructor
    public Backend(form1 form)
    {
        // Inicializo OPCServer y otros objetos OPC
        m_server = new OpcServer();
        m_server.ErrorsAsExecptions = true;
    }
}
```

3.4. Implementación de Mejoras

La implementación de mejoras en este trabajo va por la vía de la modularización del código del SCADA disponible actualmente para Huatacondo y no entra en detalles tan específicos sino que más bien ataca la estructura organizativa del programa en uno de los arhivos específicos de este.

3.4.1. Trabajo sobre el código base

Trabajo sobre form1

En el form1, como se propone más arriba, se separa el BackEnd del FrontEnd. Por simplicidad se crea otro archivo que será el FrontEnd y el form1 se transformará en el BackEnd. Para probar desempeño antes y después, se montará un servidor que permita generar entradas al programa y así simular su funcionamiento como si estuviese conectado en línea con la microrred.

3.4.2. Simulación de Servidor OPC y Entradas

Para probar el programa se corrió un servidor de prueba que pudiese conectarse al SCADA operando. Dado que no se contaban con datos en tiempo real de Huatacondo, se propuso simular datos de entrada. Para ello se tomaron las entradas del servidor y se simularon valores como se muestra en las Figuras 3.37, 3.38 y 3.39.

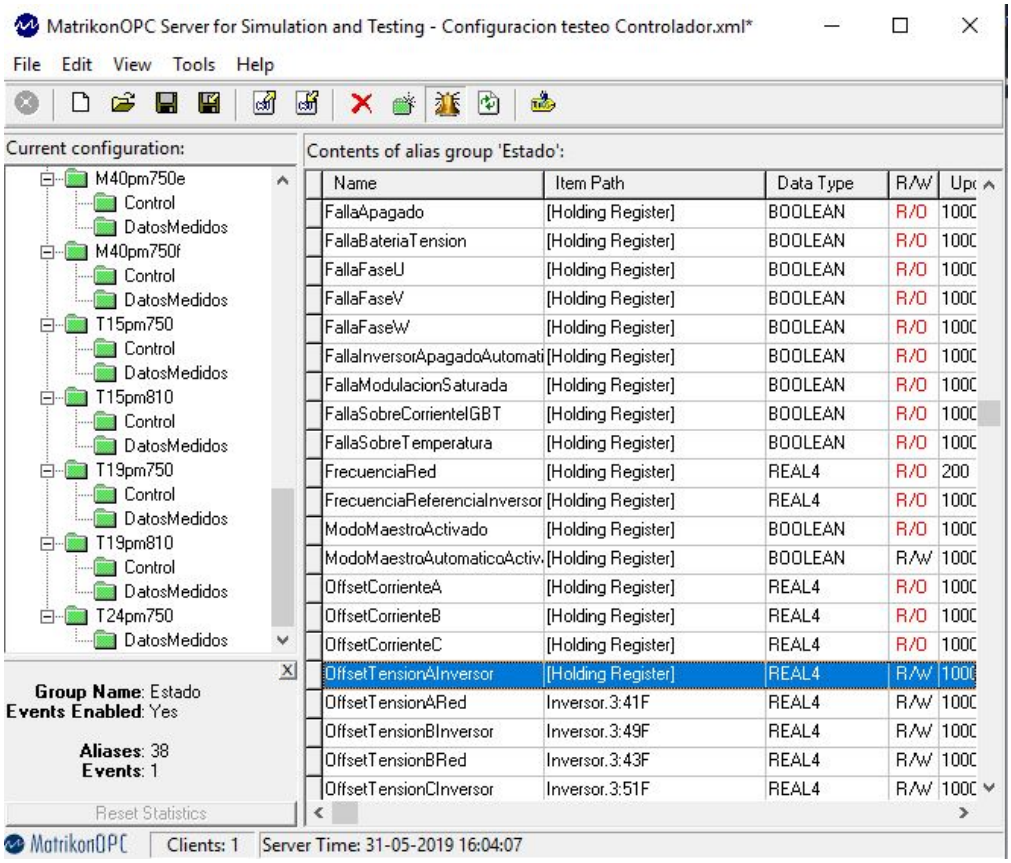


Figura 3.37: OPC Server - Configuración de Tags

Las carpetas ubicadas en el panel *"Current Configuration"* del programa *MatrikonOPC Server and Simulation* contienen los datos o *tags* que el programa ESUSCON lee de la operación de la microrred. Entre ellos se encuentran parámetros en los puntos de medición como corrientes, voltajes, frecuencias, etc. medidas en [p.u.], además de variable boolean que describen estados de falla, operación, conexión, modos activados, etc. Cada una de las variables tiene un nombre, un *"Item Path"* asociado (que es el valor), el tipo de dato (boolean, real4, etc) y si es leíble-escribible (R/W). Cada variable es llamada *tag* y para conectarlas al servidor deben ser añadidas al programa *MatrikonOPC Explorer*.

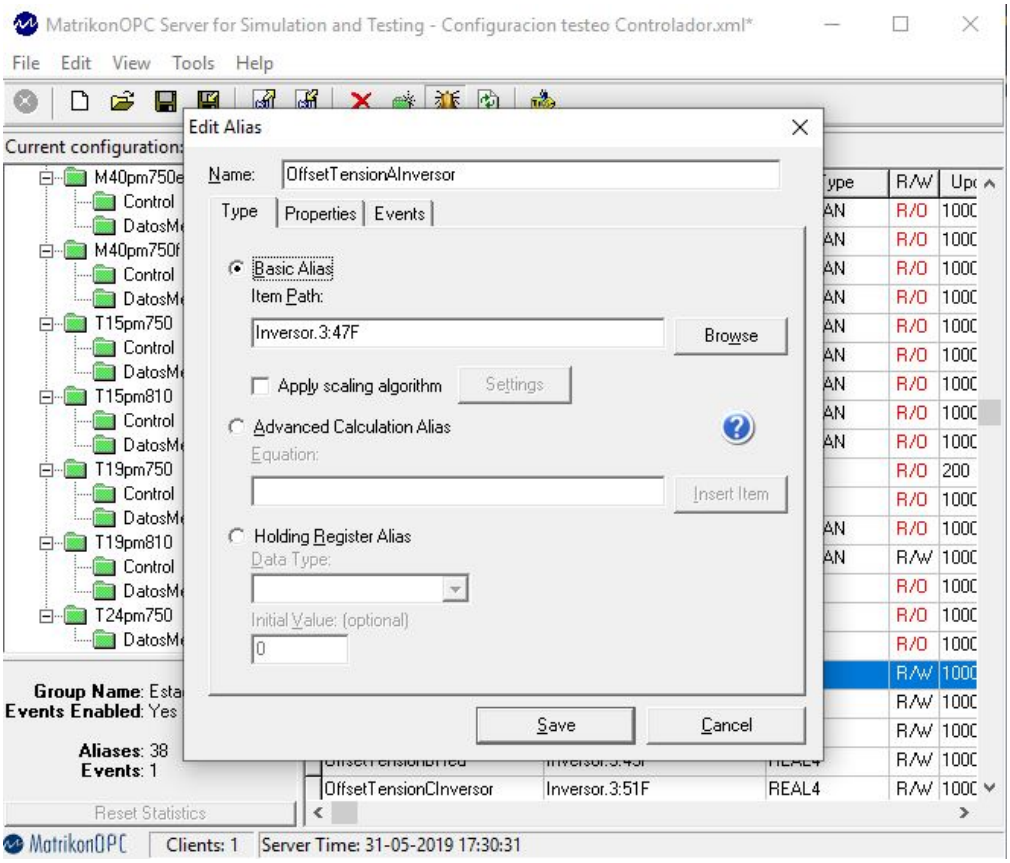


Figura 3.38: OPC Server - Configuración de Tags/Edit Alias

Al hacer doble click con el botón secundario sobre un *tag* se despliega un pequeño menú donde aparece la opción *Edit Alias*. Al presionarlo, aparece el cuadro que se visualiza en la Fig. 3.38, en el cual se puede editar el nombre y el valor del *tag* que representa una de las entradas que leerá el programa ESUSCON.

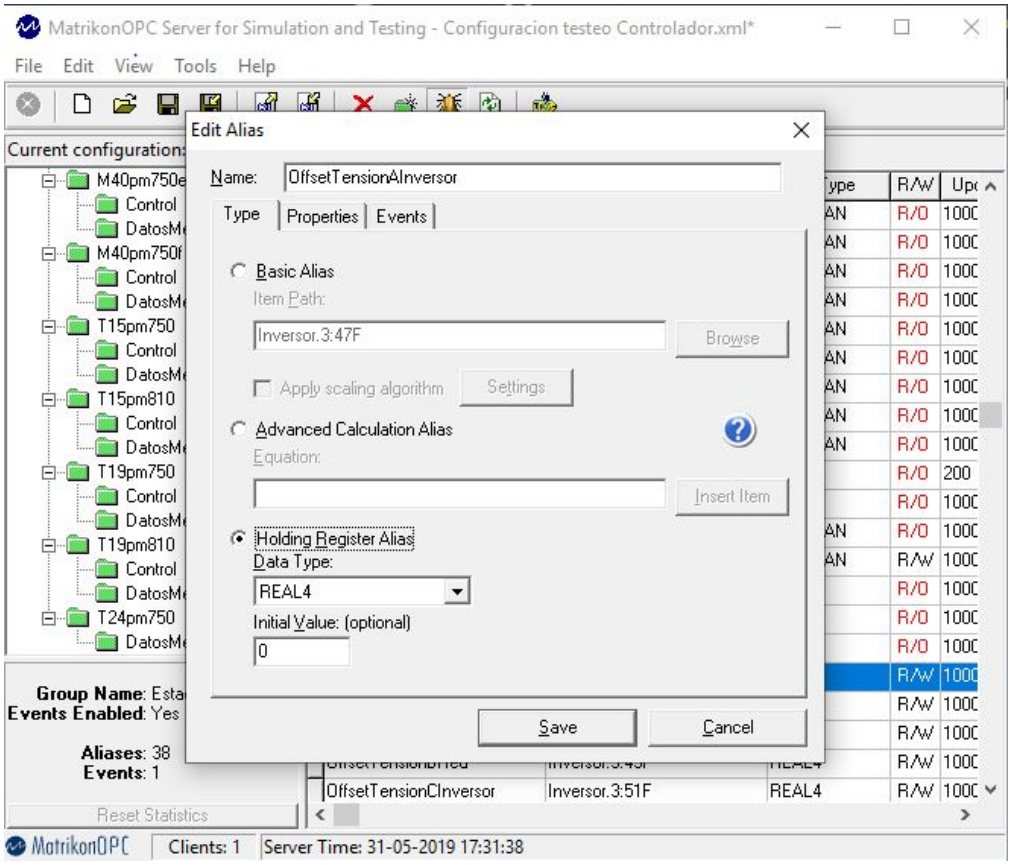


Figura 3.39: OPC Server - Configuración de Tags/Edit Alias/Holding register alias

En las Fig. 3.37, Fig. 3.38 y Fig. 3.39 muestran la configuración de los Tags que servirán de entradas de simulación para el programa SCADA.

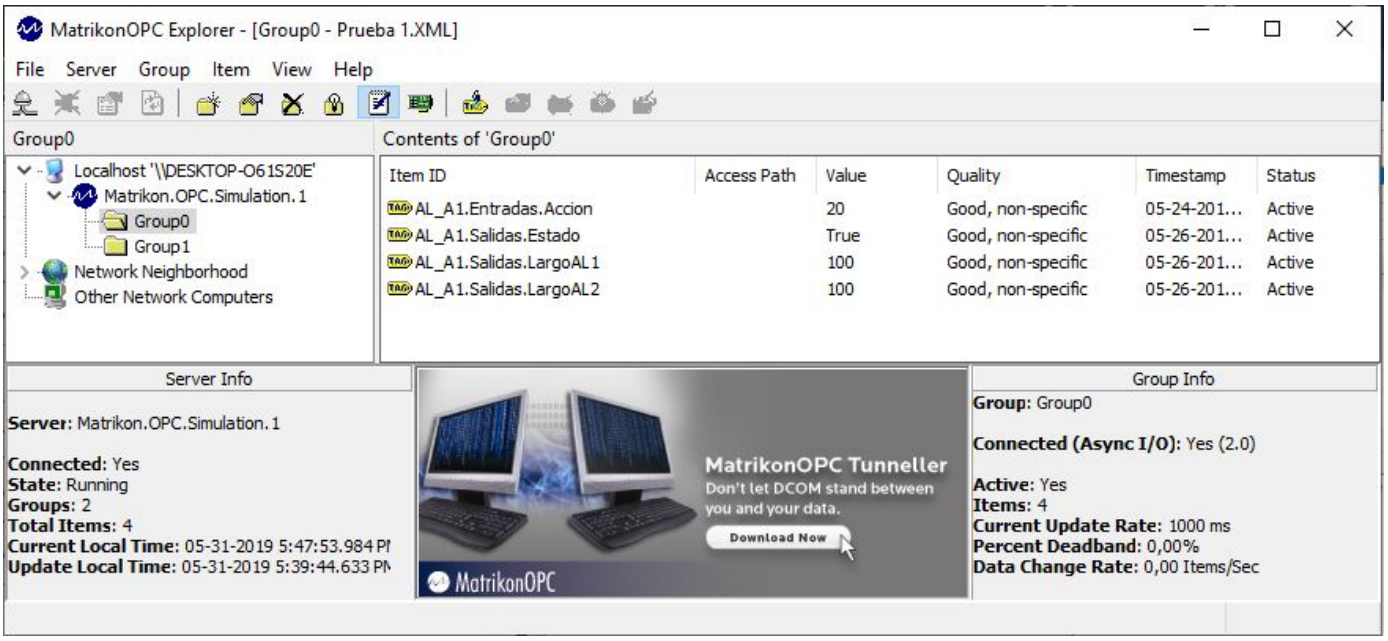


Figura 3.40: Matrikon OPC Explorer - Tags conectados

En las Fig. 3.41 y Fig. 3.41 se muestra el Explorador de Matrikon OPC y como se agregan los tags al servidor simulado. Una vez agregado se debe conectar el programa SCADA desde su ventana que se muestra en la Fig. 3.42

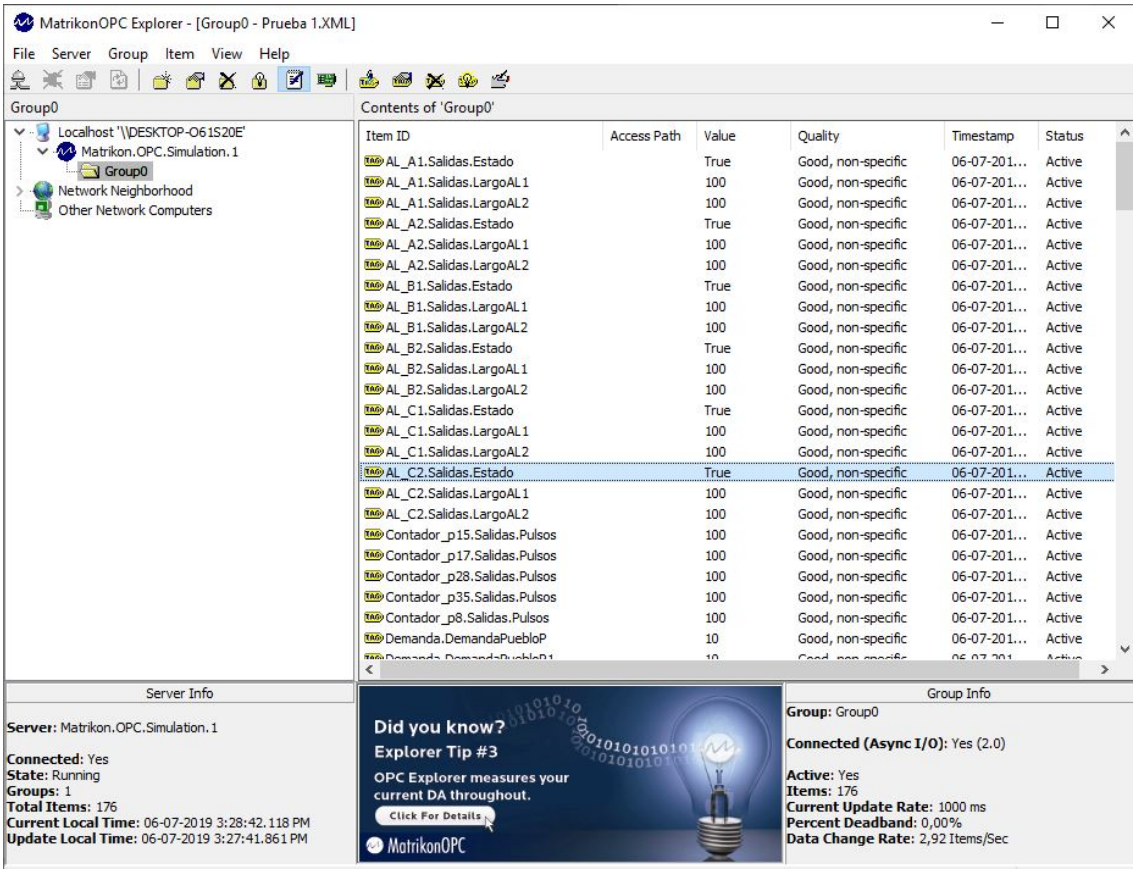


Figura 3.41: Matrikon OPC Explorer - Tags conectados completos

Una vez abierto el FrontEnd del SCADA, se debe proceder a conectar al servidor. Se selecciona el servidor disponible y se comienza la simulación.

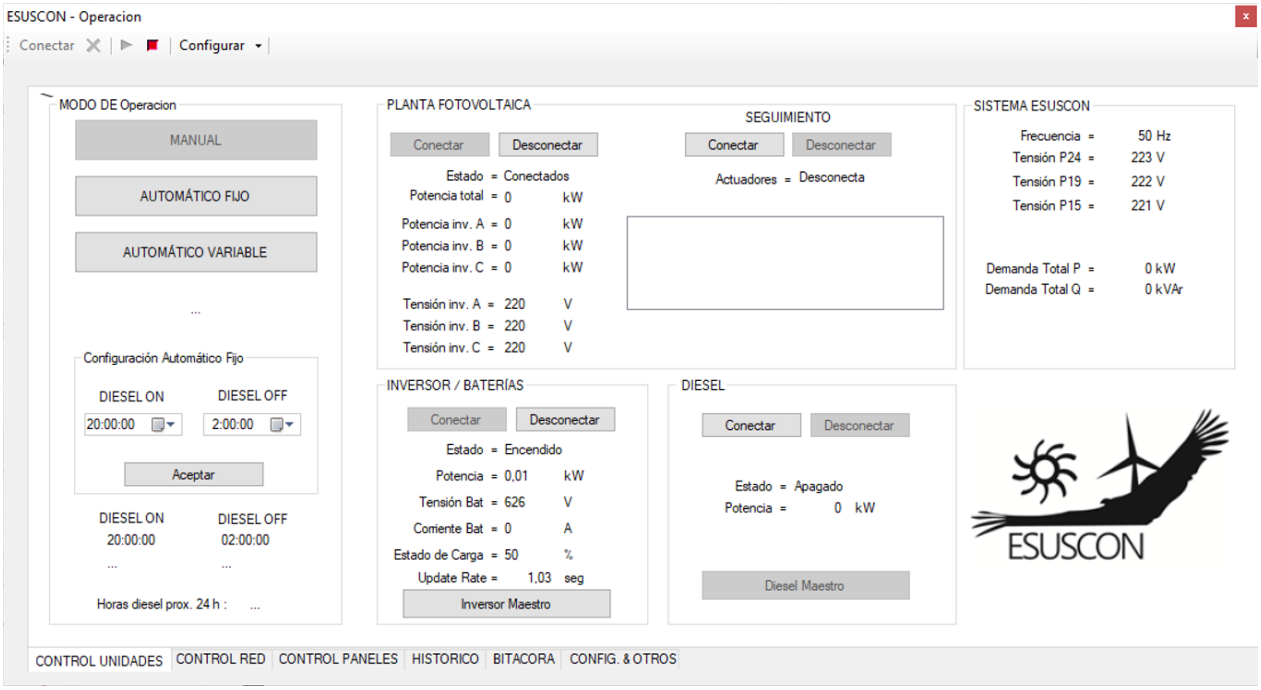


Figura 3.42: FrontEnd SCADA Huatacondo

3.4.3. Depuración, pruebas y experimentación

Para realizar las pruebas se propuso la utilización de una herramienta de profiling. En el caso del Visual Studio, en la opción *"Analyze"*, se seleccionó la opción de *"Launch Perfomance Wizard"*, herramienta para obtener datos acerca del funcionamiento del sistema. Al mismo tiempo el programa se conecto al servidor y procedió a realizar la simulación.

Pruebas

La prueba principal consiste en correr el programa junto con el servidor en dos estados:

- El programa sin implementar la modularización propuesta
- El programa una vez implementada la modularización propuesta inicialmente.

Esta modularización consiste solamente en separar el BackEnd del FrontEnd, con lo cual quedarían funciones que pertenecen a los otros módulos que se proponen más arriba en el documento *"form1"*, pero que escapan al alcance de esta memoria y que se dejan como propuestas para abordarlas en un trabajo futuro.

En paralelo, se utiliza la opción *"Launch Perfomance Wizard>Instrumentation"*. El método de *Instrumentación* genera una medida de diferentes métricas disponibles en el programa Visual Studio en la opción *"Analyze"*.

Se debe considerar también que la versión disponible se está corriendo con datos simulados en un servidor simulado y además que por simplicidad. la opción que hace funcionar el EMS se encuentra desactivada.

Métricas de Desempeño

Se estudió un set de métricas que están disponibles en la opción *Analyze* del *Microsoft Visual Estudio 2010* para las cuales la herramienta *"Launch Perfomance Wizard"* arroja resultados. Estas métricas son las ya mencionadas en el Marco Teórico:

- *Elapsed Inclusive Time values*
- *Elapsed Exclusive Time values*
- Número de Llamados

Tabla 3.1: Cuadro para evaluar los casos y el desempeño del programa

Criterios de Evaluación	Con Interacción	Sin Interacción
Con Modificación	M/I	M/SI
Sin Modificación	SM/I	SM/SI

Para evaluar el desempeño del programa, se tomaron estas métricas y se analizaron las funciones en el caso de los Inclusive y Exclusive Time Values, mientras que en el caso del número de llamadas, se tomó el casod de las funciones y los módulos.

Como se comentó anteriormente se estudiará un caso base y otro modificado. Para estudiar el caso base, se tomaron los datos que se muestran en la Fig. **??**. Ahora bien, se propone considerar el caso modificado y además estudiar dos estados:

- Sin interactuar con el SCADA
- Interactuando con el SCADA

Para evaluar estado con interacción se debe definir un protocolo sobre cuales elementos se chequearán o revisarán durante la ejecución/evaluación del programa. De esta manera la revisión de casos quedaría de la siguiente manera:

Finalmente los casos que se estudiarán son los que se muestran en la tabla 3.1:

- Sin modificar y sin interacción
- Sin modificar y con interacción
- Modificado y sin interacción
- Modificado y con interacción

Para realizar las pruebas, los casos sin interacción se correrán durante 19 min, al igual que los casos con interacción, con la diferencia que en estos últimos se realizará una rutina de manera de probar las funcionalidades del programa SCADA. Para ello, se diseño una rutina que se muestra a continuación y que indica específicamente en qué momento se debe realizar cual acción. Esta rutina toca varios casos importantes, como son:

1. Prender y apagar unidades
2. Conectar/desconectar unidades
3. Cambiar entradas de variables imitando supuestas fallas
4. Simular cambios imtempestivos en variables como potencia, voltaje y corriente, cambiándolas desde la aplicación *"MatrikonOPC Server for Simulation and Testing"*

Diseño de Rutina de Prueba de interacción

La prueba de interacción con el programa se debe realizar de manera de tocar los puntos principales del SCADA. Dada la formulación del problema y como se presenta la pantalla se seguirán los siguientes pasos:

1. Se da Run al programa (t = 0 s)
2. Se esperan 30 segundos y se conecta el servidor simulado (t = 30 s)
3. Una vez conectado se espera 1 minuto y se comienza a interactuar con el FrontEnd del SCADA(el programa se inicializa en modo *MANUAL* y por tanto es el primer modo Diesel que se prueba) (t = 1 min : 30 s)
4. Se modifican tres variables, cada una por separado durante 2 minutos cada una:
   - *T15pm810>Potencia Activa Total* pasando de 3.5 a 0.(t = 1 min : 30 s)
   - *Inversor>Estado>BateríasCorriente* pasando de 0 a 1. (t = 3 min : 30 s)
   - *M40pm750e>DatosMedidos:*
     - *PotenciaActiva1* de 0 a 1
     - *PotenciaReactiva1* de 0 a 0.3

     y *M40pm750f>Datos Medidos*:
     - *PotenciaActiva1* de 0 a 1.
     - *PotenciaActiva2* de 0 a 1.
     - *PotenciaActiva3* de 0 a 1.
     - *PotenciaActivaTotal* de 0 a 3.
     - *PotenciaReactiva1* de 0 a 0.2.
     - *PotenciaReactiva2* de 0 a 0.3.
     - *PotenciaReactiva3* de 0 a 0.2.
     - *PotenciaReactivaTotal* de 0 a 0.7.

     (t = 5 min : 30 s)
5. Se aprieta el botón:
   - Conectar Diesel (t = 7 min : 30 s)
   - Desconectar inversor (t = 9 min : 30 s)
   - En la categoría *CONTROL PANELES*, se para el *CONTROL AUTOMÁTICO* (t = 11 min : 30 s) y se *INICIA* el *CONTROL MANUAL*. (t = 12 min : 0 s):
     - Horizontal (t = 12 min : 30 s)
     - Este (t = 13 min : 0 s)
     - Oeste (t = 13 min : 30 s)
6. Cambiar hora del encendido del diesel a 2 minutos después del momento que se modificó (t = 14 min : 0 s). Se prueba el segundo modo del Diesel: Automático Fijo . Esperar 4 minutos desde que se modificó la hora. (t = 18 min : 0 s)
7. Cierre Programa (t = 19 min : 0 s)

A continuación se anexa un diagrama de flujo en la Fig. 3.43 que representa la Rutina de prueba recién mencionada.

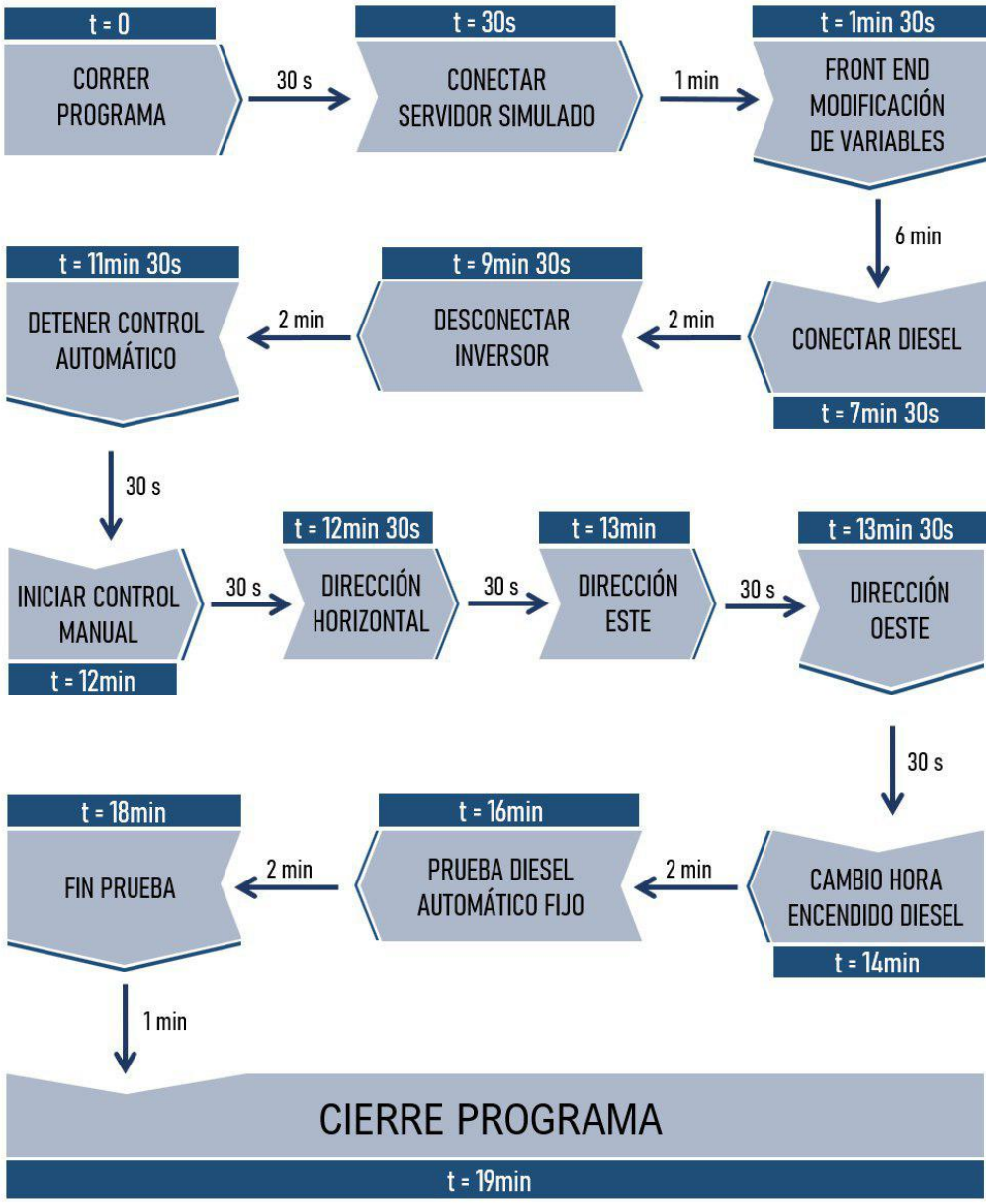


Figura 3.43: Diagrama de Flujo - Rutina de Prueba SCADA

Consideraciones

Cabe mencionar que por simplicidad para esta prueba, se consideraron los siguientes puntos:

1. No se utilizó el *Modo Automático Variable* del Diesel. Se desactivó manualmente borrando algunas líneas de código de manera que no se ejecute el EMS[1]
2. El botón de *Inversor Maestro* tampoco funciona y lanza un error cuando se aprieta, tal como se muestra en la Fig. 3.44
3. La configuración de hora de apagado/encendido de Diesel se puede realizar sin mayores problemas
4. Tampoco está disponible la función de conexión del Eólico (función "Conectar") del panel *SEGUIMIENTO*
5. El *MODO AUTOMÁTICO FIJO* del Diesel, al activarse no permite modificar otros parámetros en el SCADA
6. El *MODO AUTOMÁTICO VARIABLE* del Diesel, al activarse tampoco permite modificar otros parámetros en el SCADA.

Errores y problemas de ejecución

Al estar desactivado el EMS, el *Modo Automático Variable* no opera y por tanto, al presionar el botón, el programa intenta recuperar datos historizados que no encuentra el programa, ya que no hay, lo cual lo lleva a desconectarse, como se puede ver en la Fig. 3.44.

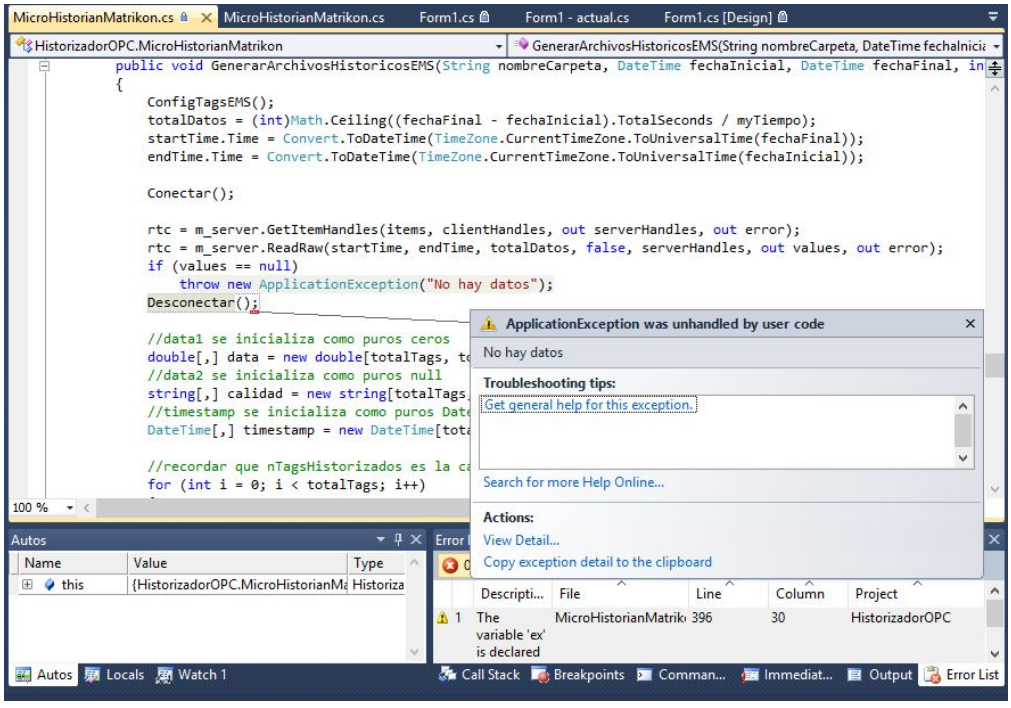


Figura 3.44: Error al conectar modo automatico variable

También al presionar el botón Inversor Maestro se produce una falla (Fig . 3.45), y se cae el programa, por tanto en la rutina anterior no se incluye esto como parte de la prueba de funcionamiento.

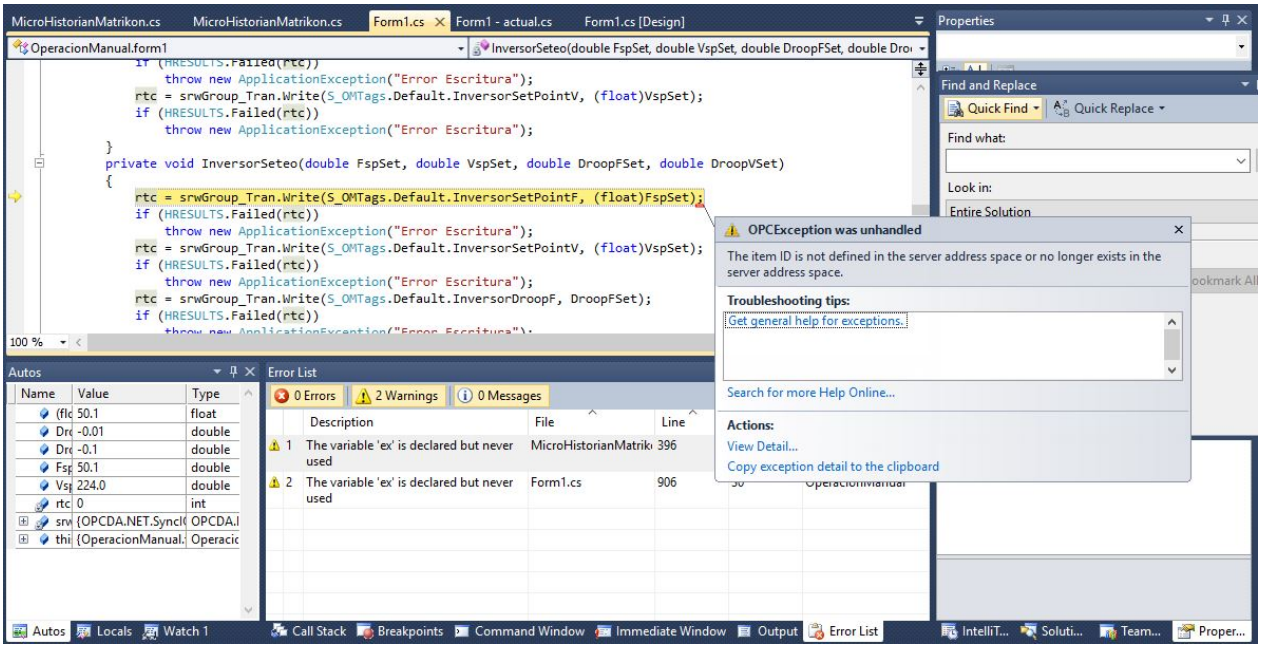


Figura 3.45: Error al presionar botón *Inversor Maestro*

Modificar Variables de Entrada: Servidor

Para realizar cambios en las variables que se simulan y lee el programa SCADA, se debe acceder a la aplicación *"MatrikonOPC Server for Simulation and Testing"* y modificar las variables disponibles.

En la Fig. 3.46 se muestra el cambio que se realiza en las variables de cambio de estado del inversor para funciones como: apagar, pasar a modo esclavo, pasar a modo maestro y sincronizar en la red.

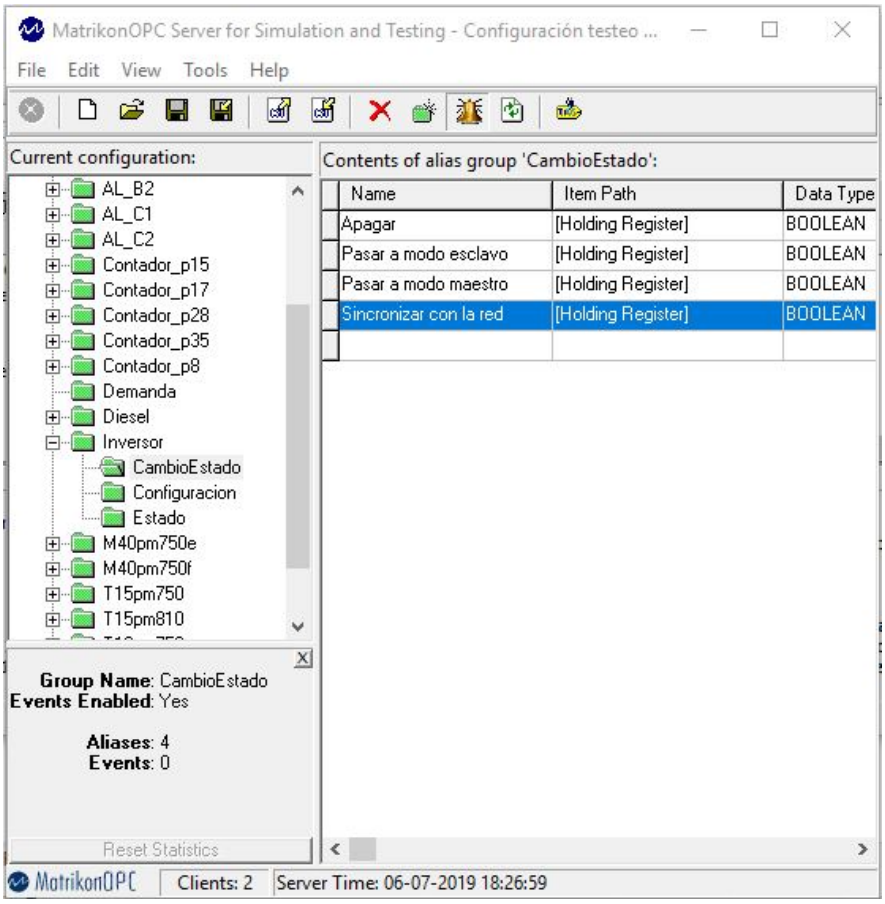


Figura 3.46: Cambio Estado Inversor - Parámetros Simulados

Capítulo 4

Resultados y análisis

4.1. Resultados

A continuación se muestra en la Fig. 4.1 como queda el programa modularizado en Frontend (*"form1"*) y Backend.

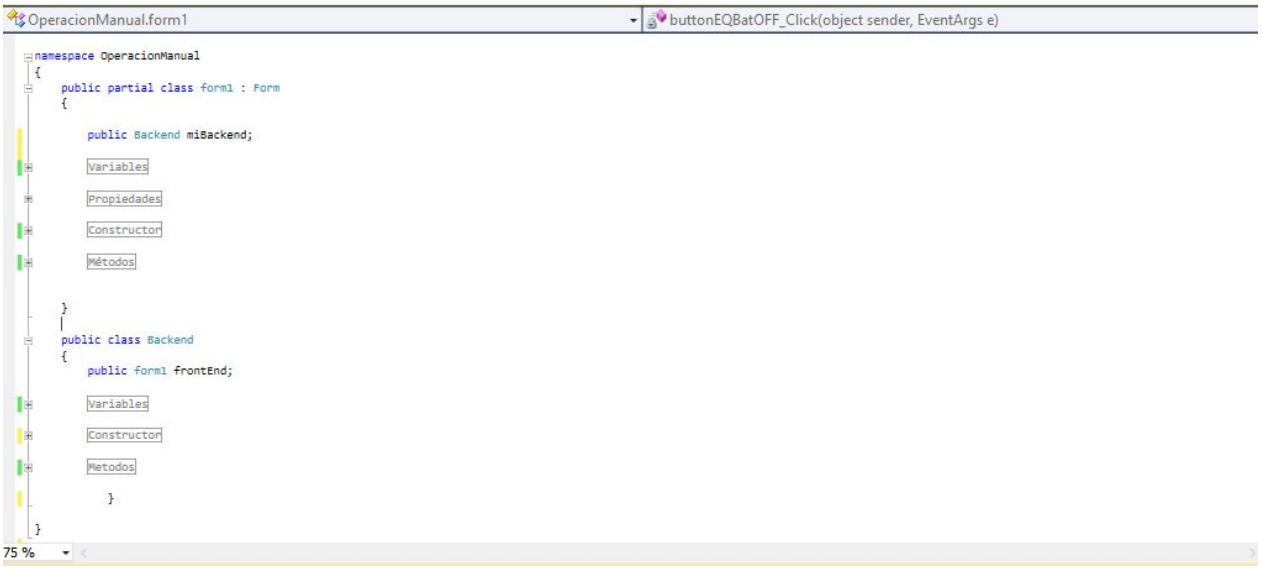


Figura 4.1: *"form1"* modularizado en Frontend y Backend

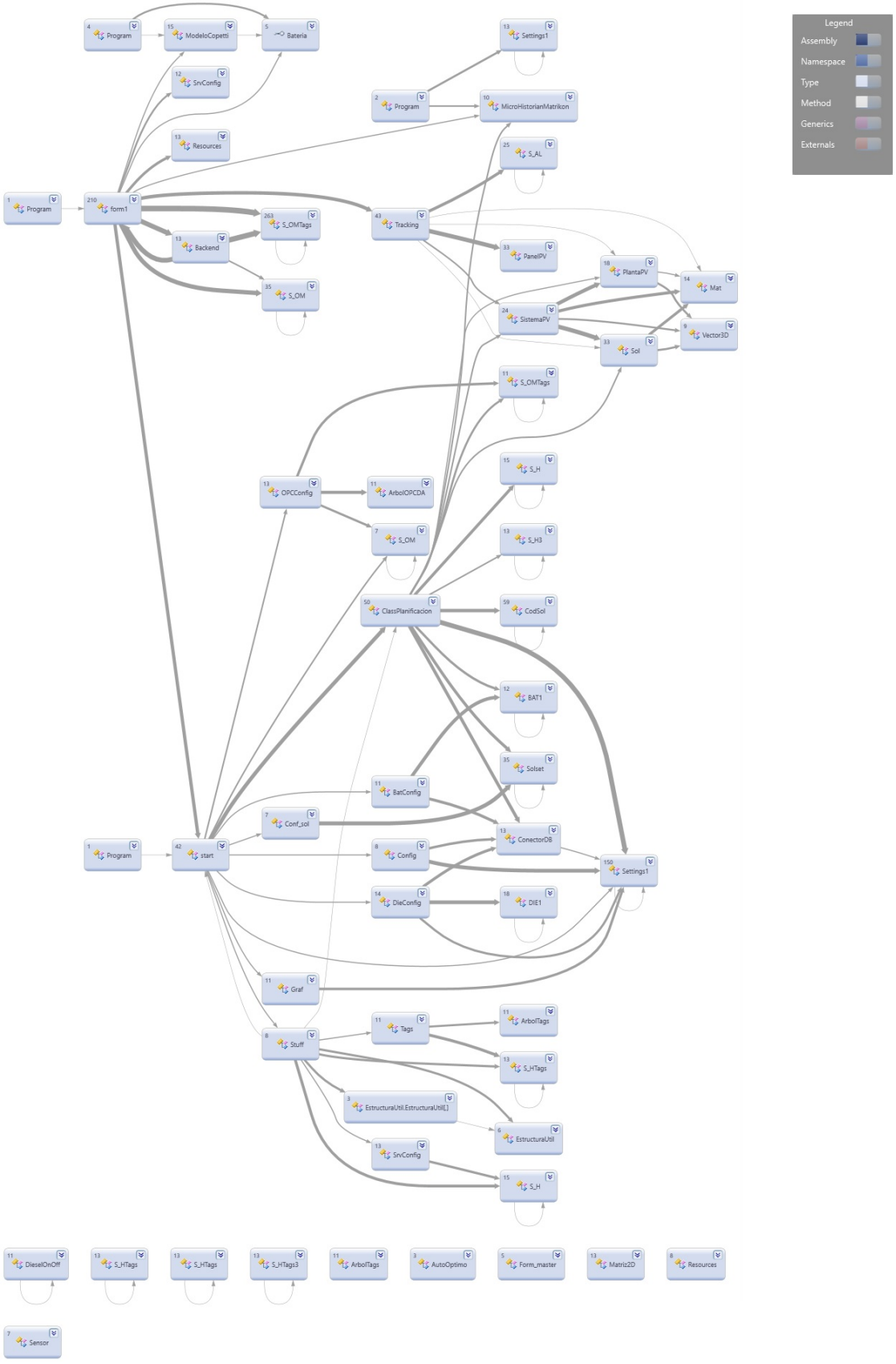


Figura 4.2: Diagrama de Clases SCADA Modularizado

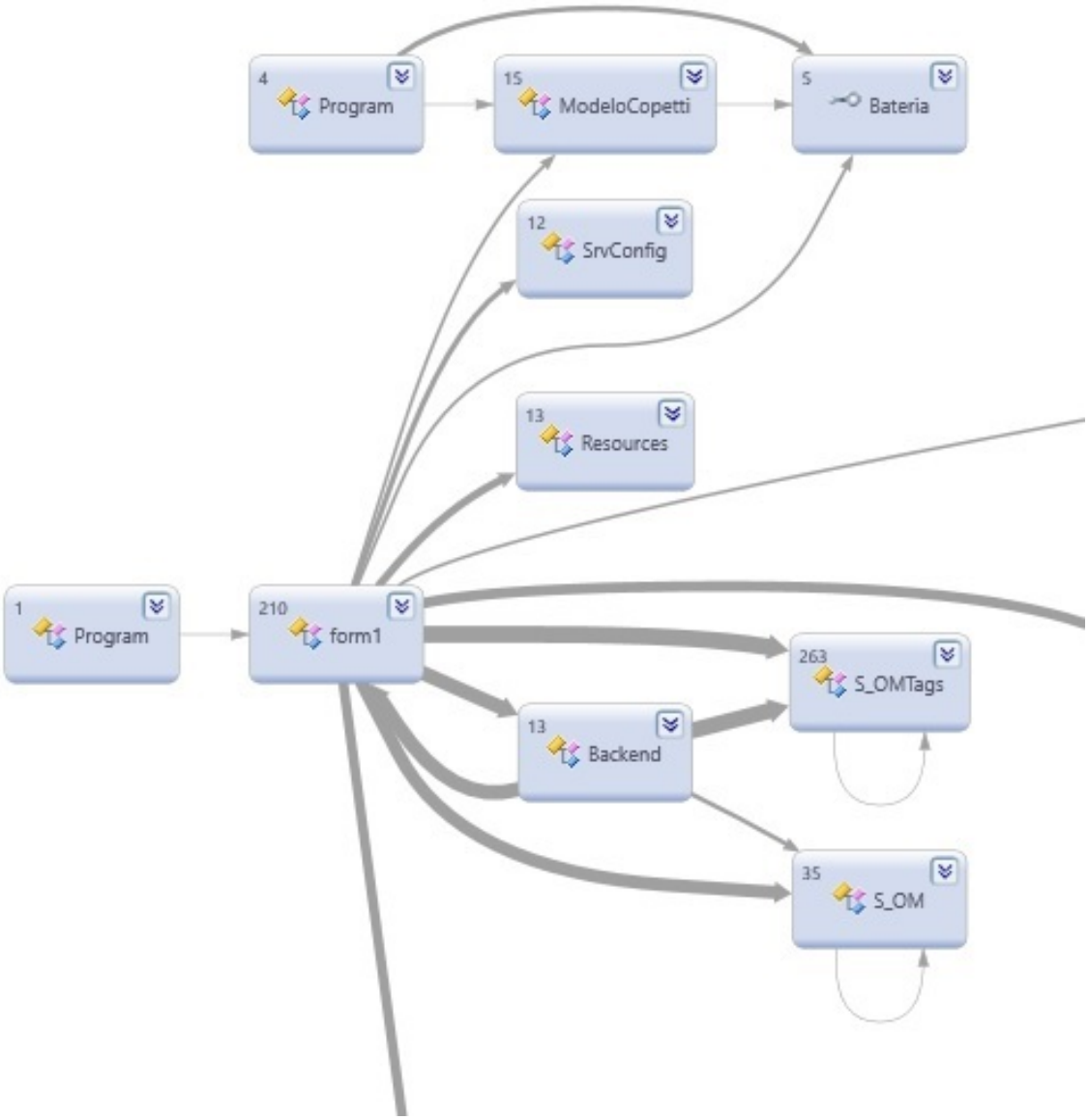


Figura 4.3: Diagrama de Clases SCADA Modularizado - Acercamiento

En la Fig. 4.3 se puede apreciar la modularización donde se separa el *form1*, que es el Frontend, del *Backend*. Cada uno contiene los elementos de los cuales se hablí en capítulos anteriores.

4.1.1. Caso 1: Programa sin Modularizar y sin interacción

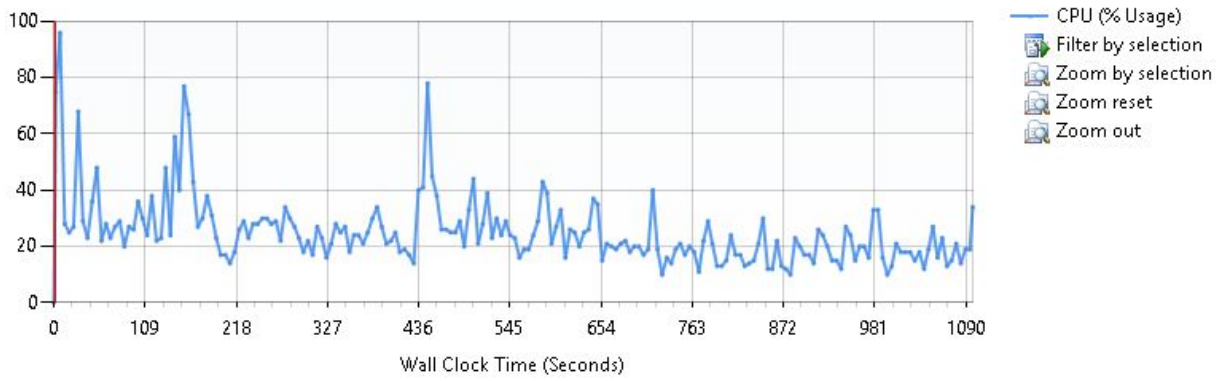


Figura 4.4: CPU Sampling del Caso SM/SI

Hot Path

The most expensive call path based on execution times

Function Name	Elapsed Inclusive Time %	Elapsed Exclusive Time %
ConsoleApplicationPruebas.exe	100,00	0,00
ConsoleApplicationPruebas.Program.Main(string[])	**100,00**	**100,00**

Related Views: Call Tree Functions

Functions With Most Individual Work

Functions with the highest exclusive application times

Name	Exclusive Time %
System.Windows.Forms.Application.Run(class System.Windows.Forms.Form)	81,39
System.Threading.Thread.Sleep(int32)	16,72
System.Configuration.SettingsBase.Save()	0,92
System.String.Concat(object,object,object)	0,24
System.Windows.Forms.Timer.Start()	0,10

Figura 4.5: Otros parámetros Caso SM/SI

4.1.2. Caso 2: Programa sin Modularizar y con interacción

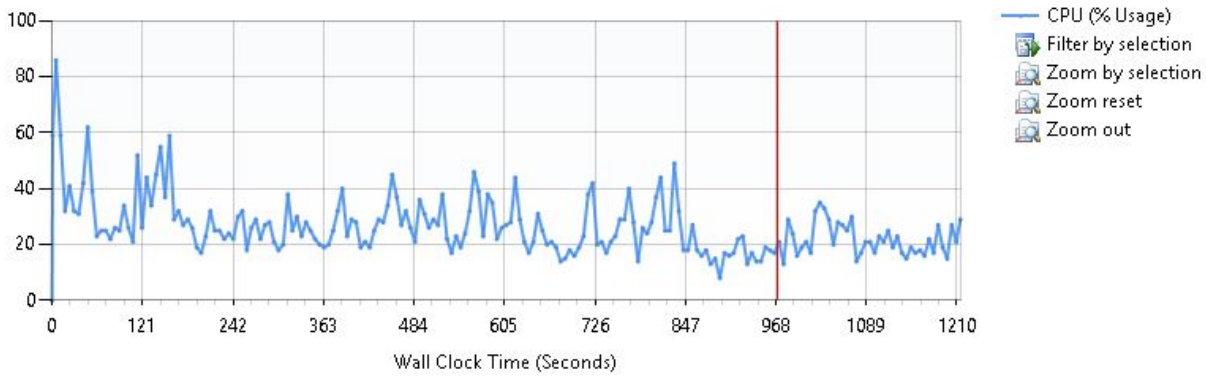


Figura 4.6: CPU Sampling del Caso SM/I

Hot Path

The most expensive call path based on execution times

Function Name	Elapsed Inclusive Time %	Elapsed Exclusive Time %
ConsoleApplicationPruebas.exe	100,00	0,00
ConsoleApplicationPruebas.Program.Main(string[])	**100,00**	**100,00**

Related Views: Call Tree Functions

Functions With Most Individual Work

Functions with the highest exclusive application times

Name	Exclusive Time %
System.Windows.Forms.Application.Run(class System.Windows.Forms.Form)	61,39
System.Threading.Thread.Sleep(int32)	37,19
System.Configuration.SettingsBase.Save()	0,77
System.String.Concat(object,object,object)	0,19
HistorizadorOPC.MicroHistorianMatrikon.GenerarArchivoHistorico(string,string,valuetype System.DateTime,va...	0,07

Figura 4.7: Otros parámetros Caso SM/I

4.1.3. Caso 3: Programa Modularizado y sin interacción

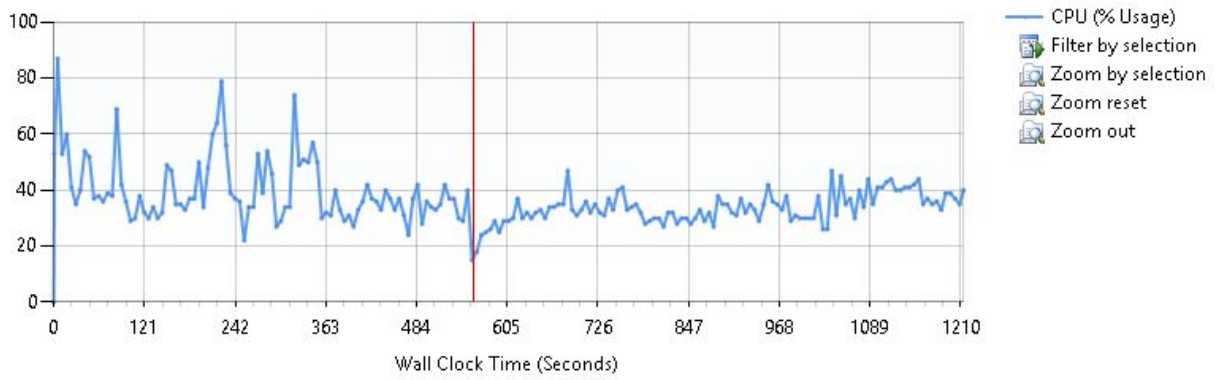


Figura 4.8: CPU Sampling del Caso M/SI

Hot Path

The most expensive call path based on execution times

Function Name	Elapsed Inclusive Time %	Elapsed Exclusive Time %
HistorizadorOPC.exe	100,00	0,00
HistorizadorOPC.Program.Main(string[])	100,00	54,33
HistorizadorOPC.MicroHistorianMatrikon.GenerarArchivoHistorico(string,strin...	**33,24**	**0,41**
HistorizadorOPC.Properties.Settings1.get_OPCServerMachine()	**11,72**	**11,72**

Related Views: Call Tree Functions

Functions With Most Individual Work

Functions with the highest exclusive application times

Name	Exclusive Time %
System.Windows.Forms.Application.Run(class System.Windows.Forms.Form)	49,33
System.Windows.Forms.Application.Run(class System.Windows.Forms.Form)	46,27
System.Threading.Thread.Sleep(int32)	2,25
System.Configuration.SettingsBase.Save()	1,19
System.String.Concat(object,object,object)	0,31

Figura 4.9: Otros parámetros Caso M/SI

4.1.4. Caso 4: Programa Modularizado y con interacción

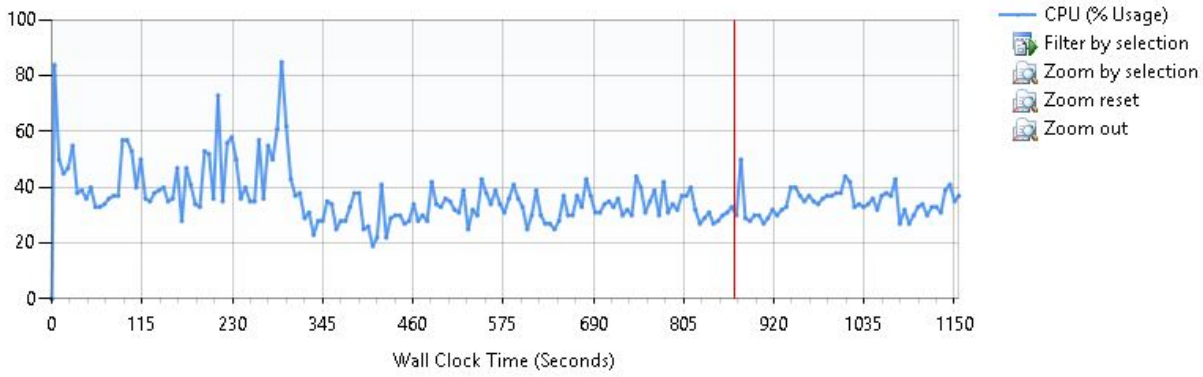


Figura 4.10: CPU Sampling del Caso M/I

Hot Path

The most expensive call path based on execution times

Function Name	Elapsed Inclusive Time %	Elapsed Exclusive Time %
HistorizadorOPC.exe	100,00	0,00
HistorizadorOPC.Program.Main(string[])	100,00	14,34
HistorizadorOPC.MicroHistorianMatrikon.GenerarArchivoHistorico(string,strin...	**77,42**	**2,06**
HistorizadorOPC.Properties.Settings1.get_OPCServerMachine()	**4,19**	**4,19**

Related Views: Call Tree Functions

Functions With Most Individual Work

Functions with the highest exclusive application times

Name	Exclusive Time %
System.Windows.Forms.Application.Run(class System.Windows.Forms.Form)	38,90
System.Windows.Forms.Application.Run(class System.Windows.Forms.Form)	37,49
System.Threading.Thread.Sleep(int32)	22,25
System.Configuration.SettingsBase.Save()	0,72
System.String.Concat(object,object,object)	0,21

Figura 4.11: Otros parámetros Caso M/I

4.1.5. Comparación de Casos

De los casos modularizados versus los no modularizados, se puede apreciar claramente que en los modularizados el trabajo se divide en 2 forms:

- En el caso 3 (M/SI), hay dos funciones llamadas *System.Windows.Forms.Application.Run(class.System.Windows.Forms.Form)* que ocupan el 49.33 % y 46.27 % respectivamente y la otra que es *System.Threading.Thread.Sleep(int32)* solo ocupa un 2.25 %.
- En el caso 4 (M/I), hay dos funciones llamadas *System.Windows.Forms.Application.Run(class.System.Windows.Forms.Form)* que ocupan el 38.9 % y 37.49 % respectivamente y la otra que es *System.Threading.Thread.Sleep(int32)* solo ocupa un 22.25 %.

Por otro lado, en los casos sin modularizar se produce solo una función form, y esta se lleva la mayor carga de trabajo:

- En el caso 1 (SM/SI), hay una funcion llamada *System.Windows.Forms.Application.Run(class.System.Windows.Forms.Form)* que ocupa el 81.39 % y la otra que es *System.Threading.Thread.Sleep(int32)* solo ocupa un 16.72 %.
- En el caso 2 (SM/I), hay una funcion llamada *System.Windows.Forms.Application.Run(class.System.Windows.Forms.Form)* que ocupa el 61.39 % y la otra que es *System.Threading.Thread.Sleep(int32)* solo ocupa un 37.19 %.

Cada una de las pruebas fue llevada a cabo durante 19 minutos, es decir, 1140 segundos, que se ve graficado en las Figuras 4.4, 4.6, 4.8 y 4.10.

Diff Report - Reportes de diferencia

El Diff report presenta una vista de la tablsa de datos. La tabla muestra el *delta* delta, o el cambio sobre la línea base. Este se calcula determinando la diferencia entre el valor antiguo, el valor de la línea base, y el valor resultante del nuevo análisis.

A continuación se realiza una comparación de los 4 casos de prueba utilizando los reportes de diferencia. A continuación cuando se refiera a caso base esta memoria, se referirá al caso sin modularizar.

Caso SM/SI v/s Caso SM/I

El caso base (SM) sin interacción (I) vs el caso base (SM) con interacción (I) presenta los siguientes reportes de diferencia:

Comparison Column	Delta	Baseline Value	Comparison Value
OperacionManual.Program.Main()	1.178.901,46	25.591,85	1.204.493,31
OperacionManual.form1.tsb_play_Click(object,class System.EventArgs)	331,97	0,00	331,97
OperacionManual.form1.tsb_conectar_Click(object,class System.EventArgs)	47,92	0,00	47,92
OperacionManual.form1.timerSOC_Tick(object,class System.EventArgs)	21.645,05	0,00	21.645,05
OperacionManual.form1.timerEstadoOPC_Tick(object,class System.EventAr	484,58	0,00	484,58
OperacionManual.form1.Substring(string,int32)	1,01	0,00	1,01
OperacionManual.form1.Stop()	40,05	2,64	42,69
OperacionManual.form1.Play()	173,40	0,00	173,40
OperacionManual.form1.InitializeComponent()	214,25	554,48	768,73
OperacionManual.form1.GuardarBitacora(class System.Windows.Forms.List	1,64	0,47	2,11
OperacionManual.form1.form1_FormClosing(object,class System.Windows	47,70	7,19	54,89
OperacionManual.form1.Dispose(bool)	43,45	42,28	85,73
OperacionManual.form1.DesconectarServidor()	3,82	0,53	4,36
OperacionManual.form1.Desconectar()	3,78	0,91	4,68
OperacionManual.form1.CrearGrupoTran()	7,44	0,00	7,44
OperacionManual.form1.CrearGrupoSoC()	2,63	0,00	2,63
OperacionManual.form1.CrearGruposMonitoreo()	70,39	0,00	70,39
OperacionManual.form1.ConectarServidor()	44,59	0,00	44,59
OperacionManual.form1.Conectar()	47,00	0,00	47,00
OperacionManual.form1.CompleteHandler2(object,class OPCDA.NET.Refre	22,40	0,00	22,40
OperacionManual.form1.CompleteHandler1(object,class OPCDA.NET.Refre	13,15	0,00	13,15
OperacionManual.form1.CerrarGrupoSoC()	5,76	0,00	5,76
OperacionManual.form1.CerrarGruposMonitoreo()	17,06	0,00	17,06
OperacionManual.form1..ctor()	416,46	984,71	1.401,17

Figura 4.12: Caso SM/SI v/s Caso SM/I - Elapsed Inclusive values, Funciones - archivo *"Operación Manual.exe"*

Comparison Column	Delta	Baseline Value	Comparison Value
OperacionManual.form1..ctor()	186,97	255,42	442,39
OperacionManual.form1.Conectar()	2,16	0,00	2,16
OperacionManual.form1.form1_FormClosing(object,class System.Windows	2,23	3,16	5,39
OperacionManual.form1.Play()	21,92	0,00	21,92
OperacionManual.form1.timerSOC_Tick(object,class System.EventArgs)	104,41	0,00	104,41
OperacionManual.form1.tsb_play_Click(object,class System.EventArgs)	1,68	0,00	1,68
OperacionManual.Program.Main()	31,11	58,25	89,36
OperacionManual.Properties.S_OM.get_DieselHoraON()	2,25	100,87	103,12
OperacionManual.Properties.S_OM.set_BatCorriente(float64)	7,66	0,00	7,66
OperacionManual.Properties.S_OM.set_BatSoC(float64)	29,10	0,00	29,10
OperacionManual.Properties.S_OM.set_BatSoC_historico(string)	43,93	0,00	43,93
OperacionManual.Properties.S_OMTags.get_InversorBateriasI()	33,27	0,00	33,27
OperacionManual.Properties.S_OMTags.get_InversorBateriasV()	23,69	0,00	23,69
OperacionManual.Properties.S_OMTags.get_InversorVmedio()	17,55	0,00	17,55

Figura 4.13: Caso SM/SI v/s Caso SM/I - Elapsed Exclusive values, Funciones - archivo *"Operación Manual.exe"*

Comparison Column	Delta	Baseline Value	Comparison Value
OperacionManual.form1.agregarEvento(object,class System.Windows.Form	18	8	26
OperacionManual.form1.CerrarGrupoTran()	1	0	1
OperacionManual.form1.CompleteHandler1(object,class OPCDA.NET.Refre	2	0	2
OperacionManual.form1.CompleteHandler2(object,class OPCDA.NET.Refre	2	0	2
OperacionManual.form1.Conectar()	1	0	1
OperacionManual.form1.ConectarServidor()	1	0	1
OperacionManual.form1.CrearGruposMonitoreo()	1	0	1
OperacionManual.form1.CrearGrupoSoC()	1	0	1
OperacionManual.form1.CrearGrupoTran()	1	0	1
OperacionManual.form1.Play()	1	0	1
OperacionManual.form1.Substring(string,int32)	4.718	1	4.719
OperacionManual.form1.timerEstadoOPC_Tick(object,class System.EventAr	239	0	239
OperacionManual.form1.timerSOC_Tick(object,class System.EventArgs)	1.179	0	1.179
OperacionManual.form1.tsb_conectar_Click(object,class System.EventArgs)	1	0	1
OperacionManual.form1.tsb_play_Click(object,class System.EventArgs)	1	0	1

Figura 4.14: Caso SM/SI v/s Caso SM/I - Número de Llamadas Funciones - archivo *"Operación Manual.exe"*

Comparison Column	Delta	Baseline Value	Comparison Value
HistorizadorOPC.dll	2	1	3
Microsoft.VisualBasic.PowerPacks.Vs.dll	11	334	345
OperacionManual.exe	13.930	67	13.997
Planificacion.dll	1.181	0	1.181
Planificacion.exe	2.380	35	2.415
Planificacion.exe	-14	35	21
SistemaFotovoltaico.dll	3	3	6

Figura 4.15: Caso SM/SI v/s Caso SM/I - Número de Llamadas Módulos - archivo *"Operación Manual.exe"*

Caso SM/SI v/s Caso M/SI

El caso base (SM) sin interacción (SI) vs el caso modularizado (M) sin interacción (I) presenta los siguientes reportes de diferencia:

Comparison Column	Delta	Baseline Value	Comparison Value
OperacionManual.form1.timerSOC_Tick(object,class System.EventArgs)	43.182,63	0,00	43.182,63
OperacionManual.form1.timerEstadoOPC_Tick(object,class System.EventArgs)	1.196,70	0,00	1.196,70
OperacionManual.form1.Stop()	121,57	2,64	124,21
OperacionManual.form1.Play()	196,29	0,00	196,29
OperacionManual.form1.InitializeComponent()	397,15	554,48	951,63
OperacionManual.form1.GuardarBitacora(class System.Windows.Forms.ListBox)	3,78	0,47	4,25
OperacionManual.form1.form1_FormClosing(object,class System.Windows.Forms.FormClosi	26,21	7,19	33,40
OperacionManual.form1.Dispose(bool)	97,10	42,28	139,38
OperacionManual.form1.Desconectar()	12,81	0,91	13,72
OperacionManual.form1.CrearGrupoTran()	10,84	0,00	10,84
OperacionManual.form1.CrearGrupoSoCFront()	3,79	0,00	3,79
OperacionManual.form1.CrearGruposMonitoreoFront()	91,72	0,00	91,72
OperacionManual.form1.Conectar()	179,36	0,00	179,36
OperacionManual.form1.CerrarGrupoTran()	1,20	0,00	1,20
OperacionManual.form1.CerrarGrupoSoCFront()	4,53	0,00	4,53
OperacionManual.form1.CerrarGruposMonitoreoFront()	15,07	0,00	15,07
OperacionManual.form1.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	4,10	1,48	5,58
OperacionManual.form1..ctor()	410,64	984,71	1.395,34
OperacionManual.Backend.Substring(string,int32)	4,28	0,00	4,28
OperacionManual.Backend.DesconectarServidor()	13,50	0,00	13,50
OperacionManual.Backend.CrearGrupoSoC()	3,79	0,00	3,79
OperacionManual.Backend.CrearGruposMonitoreo()	91,71	0,00	91,71
OperacionManual.Backend.ConectarServidor()	179,21	0,00	179,21
OperacionManual.Backend.CompleteHandler2(object,class OPCDA.NET.RefreshEventArgum	33,43	0,00	33,43
OperacionManual.Backend.CompleteHandler1(object,class OPCDA.NET.RefreshEventArgum	22,65	0,00	22,65
OperacionManual.Backend.CerrarGrupoSoC()	4,53	0,00	4,53
OperacionManual.Backend.CerrarGruposMonitoreo()	15,07	0,00	15,07

Figura 4.16: Caso SM/SI v/s Caso M/SI - Elapsed Inclusive values, Funciones - archivo *"Operación Manual.exe"*

Comparison Column	Delta	Baseline Value	Comparison Value
OperacionManual.Backend.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	2,30	0,00	2,30
OperacionManual.Backend.ConectarServidor()	3,20	0,00	3,20
OperacionManual.Backend.CrearGruposMonitoreo()	10,21	0,00	10,21
OperacionManual.Backend.DesconectarServidor()	2,66	0,00	2,66
OperacionManual.Backend.Substring(string,int32)	2,83	0,00	2,83
OperacionManual.form1..ctor()	-76,65	255,42	178,78
OperacionManual.form1.form1_FormClosing(object,class System.Windows.Forms.FormClosi	10,82	3,16	13,98
OperacionManual.form1.Play()	6,10	0,00	6,10
OperacionManual.form1.timerEstadoOPC_Tick(object,class System.EventArgs)	1,36	0,00	1,36
OperacionManual.form1.timerSOC_Tick(object,class System.EventArgs)	366,95	0,00	366,95
OperacionManual.form1.tsb_conectar_Click(object,class System.EventArgs)	2,77	0,00	2,77
OperacionManual.form1.tsb_play_Click(object,class System.EventArgs)	1,64	0,00	1,64
OperacionManual.form1.tsb_stop_Click(object,class System.EventArgs)	13,17	0,00	13,17
OperacionManual.Program.Main()	2,15	58,25	60,40
OperacionManual.Properties.S_OM..cctor()	2,18	0,31	2,49
OperacionManual.Properties.S_OM.get_DieselHoraON()	44,11	100,87	144,98
OperacionManual.Properties.S_OM.set_BatCorriente(float64)	16,45	0,00	16,45
OperacionManual.Properties.S_OM.set_BatSoC(float64)	52,64	0,00	52,64
OperacionManual.Properties.S_OM.set_BatSoC_historico(string)	70,09	0,00	70,09
OperacionManual.Properties.S_OMTags.get_InversorBateriasI()	64,35	0,00	64,35
OperacionManual.Properties.S_OMTags.get_InversorBateriasV()	45,88	0,00	45,88
OperacionManual.Properties.S_OMTags.get_InversorVmedio()	19,73	0,00	19,73

Figura 4.17: Caso SM/SI v/s Caso M/SI - Elapsed Exclusive values, Funciones - archivo *"Operación Manual.exe"*

Comparison Column	Delta	Baseline Value	Comparison Value
OperacionManual.form1.DesconectarServidor()	-1	1	0
OperacionManual.form1.CrearGrupoTran()	1	0	1
OperacionManual.form1.CrearGrupoSoCFront()	1	0	1
OperacionManual.form1.CrearGruposMonitoreoFront()	1	0	1
OperacionManual.form1.Conexion()	12	0	12
OperacionManual.form1.Conectar()	1	0	1
OperacionManual.form1.CerrarGrupoTran()	1	0	1
OperacionManual.form1.CerrarGrupoSoCFront()	2	0	2
OperacionManual.form1.CerrarGrupoSoC()	-1	1	0
OperacionManual.form1.CerrarGruposMonitoreoFront()	2	0	2
OperacionManual.form1.CerrarGruposMonitoreo()	-1	1	0
OperacionManual.form1.buttonredt19pm810()	1	0	1
OperacionManual.form1.buttonredt19pm750()	1	0	1
OperacionManual.form1.buttonredt15pm810()	1	0	1
OperacionManual.form1.buttonredt15pm750()	1	0	1
OperacionManual.form1.buttonimaestro()	1	0	1
OperacionManual.form1.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	13	8	21
OperacionManual.Backend.Substring(string,int32)	4.419	0	4.419
OperacionManual.Backend.DesconectarServidor()	1	0	1
OperacionManual.Backend.CrearGrupoSoC()	1	0	1
OperacionManual.Backend.CrearGruposMonitoreo()	1	0	1
OperacionManual.Backend.ConectarServidor()	1	0	1
OperacionManual.Backend.CompleteHandler2(object,class OPCDA.NET.RefreshEventArgum	4	0	4
OperacionManual.Backend.CompleteHandler1(object,class OPCDA.NET.RefreshEventArgum	2	0	2
OperacionManual.Backend.CerrarGrupoSoC()	2	0	2
OperacionManual.Backend.CerrarGruposMonitoreo()	2	0	2

Figura 4.18: Caso SM/SI v/s Caso M/SI - Número de Llamadas Funciones - archivo *"Operación Manual.exe"*

Comparison Column		Delta	Baseline Value	Comparison Value
SistemaFotovoltaico.dll	↑	3	3	6
Planificacion.exe	↑	2.230	35	2.265
Planificacion.exe	↓	-11	35	24
Planificacion.dll	↑	1.106	0	1.106
OperacionManual.exe	↑	13.290	67	13.357
Microsoft.VisualBasic.PowerPacks.Vs.dll	↑	11	334	345
HistorizadorOPC.dll	↑	2	1	3

Figura 4.19: Caso SM/SI v/s Caso M/SI - Número de Llamadas Módulos - archivo *"Operación Manual.exe"*

Caso SM/I v/s Caso M/I

El caso base (SM) con interacción (I) vs el caso modularizado (M) con interacción (I) presenta los siguientes reportes de diferencia:

Comparison Column		Delta	Baseline Value	Comparison Value
OperacionManual.form1.timerEstadoOPC_Tick(object,class System.EventArgs)	↑	459,69	484,58	944,27
OperacionManual.form1.TestGrupoTran()	↑	4,89	0,00	4,89
OperacionManual.form1.Substring(string,int32)	↓	-1,01	1,01	0,00
OperacionManual.form1.Stop()	↓	-16,03	42,69	26,66
OperacionManual.form1.Play()	↓	-31,14	173,40	142,26
OperacionManual.form1.inversor_OFF_Click(object,class System.EventArgs)	↑	42,66	0,00	42,66
OperacionManual.form1.InitializeComponent()	↓	-227,49	768,73	541,24
OperacionManual.form1.form1_FormClosing(object,class System.Windows.Forms.FormClosingEventArgs)	↓	-22,07	54,89	32,82
OperacionManual.form1.FinProceso()	↑	3,94	0,00	3,94
OperacionManual.form1.Dispose(bool)	↑	25,93	85,73	111,66
OperacionManual.form1.diesel_ON_Click(object,class System.EventArgs)	↑	22,01	0,00	22,01
OperacionManual.form1.DesconectarServidor()	↓	-4,36	4,36	0,00
OperacionManual.form1.Desconectar()	↑	9,14	4,68	13,83
OperacionManual.form1.CrearGrupoTran()	↑	48,27	7,44	55,71
OperacionManual.form1.CrearGrupoSoCFront()	↑	3,33	0,00	3,33
OperacionManual.form1.CrearGrupoSoC()	↓	-2,63	2,63	0,00
OperacionManual.form1.CrearGruposMonitoreoFront()	↑	68,52	0,00	68,52
OperacionManual.form1.CrearGruposMonitoreo()	↓	-70,39	70,39	0,00
OperacionManual.form1.ConectarServidor()	↓	-44,59	44,59	0,00
OperacionManual.form1.Conectar()	↑	31,82	47,00	78,82
OperacionManual.form1.CompleteHandler2(object,class OPCDA.NET.RefreshEventArguments)	↓	-22,40	22,40	0,00
OperacionManual.form1.CompleteHandler1(object,class OPCDA.NET.RefreshEventArguments)	↓	-13,15	13,15	0,00
OperacionManual.form1.CerrarGrupoTran()	↑	2,80	0,74	3,54
OperacionManual.form1.CerrarGrupoSoCFront()	↑	2,66	0,00	2,66
OperacionManual.form1.CerrarGrupoSoC()	↓	-5,76	5,76	0,00
OperacionManual.form1.CerrarGruposMonitoreoFront()	↑	7,16	0,00	7,16
OperacionManual.form1.CerrarGruposMonitoreo()	↓	-17,06	17,06	0,00

Figura 4.20: Caso SM/I v/s Caso M/I - Elapsed Inclusive values, Funciones - archivo *"Operación Manual.exe"*

Comparison Column		Delta	Baseline Value	Comparison Value
OperacionManual.Program.Main()	↑	19,99	89,36	109,34
OperacionManual.form1.tsb_stop_Click(object,class System.EventArgs)	↑	4,04	0,00	4,04
OperacionManual.form1.timerTestGrp_Tick(object,class System.EventArgs)	↑	2,55	0,00	2,55
OperacionManual.form1.timerSOC_Tick(object,class System.EventArgs)	↑	174,70	104,41	279,11
OperacionManual.form1.Play()	↓	-15,19	21,92	6,73
OperacionManual.form1.form1_FormClosing(object,class System.Windows.Forms.FormClosingEventArgs)	↑	5,69	5,39	11,07
OperacionManual.form1.Conectar()	↓	-2,14	2,16	0,02
OperacionManual.form1..ctor()	↓	-165,85	442,39	276,54
OperacionManual.Backend.Substring(string,int32)	↑	2,45	0,00	2,45
OperacionManual.Backend.DesconectarServidor()	↑	1,89	0,00	1,89
OperacionManual.Backend.CrearGruposMonitoreo()	↑	8,20	0,00	8,20
OperacionManual.Backend.ConectarServidor()	↑	3,01	0,00	3,01
OperacionManual.Backend.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	↑	1,68	0,00	1,68

Figura 4.21: Caso SM/I v/s Caso M/I - Elapsed Exclusive values, Funciones - archivo *"Operación Manual.exe"*

Comparison Column		Delta	Baseline Value	Comparison Value
OperacionManual.form1.Conexion()	↑	12	0	12
OperacionManual.form1.ConectarServidor()	↓	-1	1	0
OperacionManual.form1.CompleteHandler2(object,class OPCDA.NET.RefreshEventArguments)	↓	-2	2	0
OperacionManual.form1.CompleteHandler1(object,class OPCDA.NET.RefreshEventArguments)	↓	-2	2	0
OperacionManual.form1.CerrarGrupoTran()	↑	3	1	4
OperacionManual.form1.CerrarGrupoSoCFront()	↑	2	0	2
OperacionManual.form1.CerrarGrupoSoC()	↓	-1	1	0
OperacionManual.form1.CerrarGruposMonitoreoFront()	↑	2	0	2
OperacionManual.form1.CerrarGruposMonitoreo()	↓	-1	1	0
OperacionManual.form1.buttonredt19pm810()	↑	7	0	7
OperacionManual.form1.buttonredt19pm750()	↑	7	0	7
OperacionManual.form1.buttonredt15pm810()	↑	7	0	7
OperacionManual.form1.buttonredt15pm750()	↑	7	0	7
OperacionManual.form1.buttonModoAutoConfigOK_Click(object,class System.EventArgs)	↑	1	0	1
OperacionManual.form1.buttonimaestro()	↑	7	0	7
OperacionManual.form1.buttonAutoFijo_Click(object,class System.EventArgs)	↑	1	0	1
OperacionManual.form1.button_TrackingPV_ManualON_Click(object,class System.EventArgs)	↑	4	0	4
OperacionManual.form1.button_TrackingPV_ManualOFF_Click(object,class System.EventArgs)	↑	4	0	4
OperacionManual.form1.button_TrackingPV_AutoOFF_Click(object,class System.EventArgs)	↑	1	0	1
OperacionManual.form1.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	↑	326	26	352
OperacionManual.Backend.Substring(string,int32)	↑	4.331	0	4.331
OperacionManual.Backend.DesconectarServidor()	↑	1	0	1
OperacionManual.Backend.CrearGrupoSoC()	↑	1	0	1
OperacionManual.Backend.CrearGruposMonitoreo()	↑	1	0	1
OperacionManual.Backend.ConectarServidor()	↑	1	0	1
OperacionManual.Backend.CompleteHandler2(object,class OPCDA.NET.RefreshEventArguments)	↑	14	0	14
OperacionManual.Backend.CompleteHandler1(object,class OPCDA.NET.RefreshEventArguments)	↑	46	0	46

Figura 4.22: Caso SM/I v/s Caso M/I - Número de Llamadas Funciones - archivo *"Operación Manual.exe"*

Comparison Column		Delta	Baseline Value	Comparison Value
Planificacion.dll	↓	-99	1.181	1.082
OperacionManual.exe	↑	2.409	13.997	16.406
Microsoft.VisualBasic.PowerPacks.Vs.dll	↑	66	345	411

Figura 4.23: Caso SM/I v/s Caso M/I - Número de Llamadas Módulos - archivo *"Operación Manual.exe"*

Caso M/SI v/s Caso M/I

El caso modularizado (M) sin interacción (I) vs el caso modularizado (M) con interacción (I) presenta los siguientes reportes de diferencia:

Comparison Column		Delta	Baseline Value	Comparison Value
OperacionManual.form1.InitializeComponent()	↓	-410,39	951,63	541,24
OperacionManual.form1.GuardarBitacora(class System.Windows.Forms.ListBox)	↓	-1,79	4,25	2,46
OperacionManual.form1.FinProceso()	↑	3,94	0,00	3,94
OperacionManual.form1.Dispose(bool)	↓	-27,72	139,38	111,66
OperacionManual.form1.diesel_ON_Click(object,class System.EventArgs)	↑	22,01	0,00	22,01
OperacionManual.form1.CrearGrupoTran()	↑	44,87	10,84	55,71
OperacionManual.form1.CrearGruposMonitoreoFront()	↓	-23,20	91,72	68,52
OperacionManual.form1.Conectar()	↓	-100,53	179,36	78,82
OperacionManual.form1.CerrarGrupoTran()	↑	2,34	1,20	3,54
OperacionManual.form1.CerrarGrupoSoCFront()	↓	-1,87	4,53	2,66
OperacionManual.form1.CerrarGruposMonitoreoFront()	↓	-7,91	15,07	7,16
OperacionManual.form1.buttonModoAutoConfigOK_Click(object,class System.EventArgs)	↑	18,82	0,00	18,82
OperacionManual.form1.buttonAutoFijo_Click(object,class System.EventArgs)	↑	27,42	0,00	27,42
OperacionManual.form1.button_TrackingPV_ManualON_Click(object,class System.EventArgs)	↑	198,92	0,00	198,92
OperacionManual.form1.button_TrackingPV_ManualOFF_Click(object,class System.EventArgs)	↑	136,95	0,00	136,95
OperacionManual.form1.button_TrackingPV_AutoOFF_Click(object,class System.EventArgs)	↑	18,77	0,00	18,77
OperacionManual.form1.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	↑	11,71	5,58	17,29
OperacionManual.form1..ctor()	↓	-331,61	1.395,34	1.063,74
OperacionManual.Backend.DesconectarServidor()	↓	-4,04	13,50	9,46
OperacionManual.Backend.CrearGruposMonitoreo()	↓	-23,20	91,71	68,52
OperacionManual.Backend.ConectarServidor()	↓	-100,85	179,21	78,36
OperacionManual.Backend.CompleteHandler2(object,class OPCDA.NET.RefreshEventArguments)	↑	65,10	33,43	98,53
OperacionManual.Backend.CompleteHandler1(object,class OPCDA.NET.RefreshEventArguments)	↑	16,04	22,65	38,69
OperacionManual.Backend.CerrarGrupoSoC()	↓	-1,87	4,53	2,66
OperacionManual.Backend.CerrarGruposMonitoreo()	↓	-7,91	15,07	7,16
OperacionManual.Backend.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	↑	11,55	6,04	17,59
OperacionManual.Backend..ctor(class OperacionManual.form1)	↑	2,09	28,82	30,91

Figura 4.24: Caso M/SI v/s Caso M/I - Elapsed Inclusive values, Funciones - archivo *"Operación Manual.exe"*

Comparison Column		Delta	Baseline Value	Comparison Value
OperacionManual.Properties.S_OMTags.get_InversorVmedio()	↓	-6,52	19,73	13,21
OperacionManual.Properties.S_OMTags.get_InversorEncendido()	↑	1,29	0,40	1,69
OperacionManual.Properties.S_OMTags.get_InversorBateriasV()	↓	-6,97	45,88	38,90
OperacionManual.Properties.S_OMTags.get_InversorBateriasI()	↓	-14,15	64,35	50,20
OperacionManual.Properties.S_OMTags.get_GenDieselisEstadoSincronizando()	↑	1,39	0,42	1,81
OperacionManual.Properties.S_OMTags.get_GenDieselisEstadoIdle()	↑	1,04	0,30	1,35
OperacionManual.Properties.S_OMTags.get_GenDieselisEstadoCoolDown()	↑	1,10	0,31	1,41
OperacionManual.Properties.S_OMTags.get_GenDieselisEstadoConectado()	↑	1,15	0,46	1,61
OperacionManual.Properties.S_OMTags.get_GenDieselisEstadoApagado()	↑	1,21	0,32	1,53
OperacionManual.Properties.S_OM.set_BatSoC_historico(string)	↓	-11,17	70,09	58,92
OperacionManual.Properties.S_OM.set_BatSoC(float64)	↓	-11,50	52,64	41,14
OperacionManual.Properties.S_OM.set_BatCorriente(float64)	↓	-3,50	16,45	12,94
OperacionManual.Properties.S_OM.get_DieselHoraON()	↓	-3,84	144,98	141,15
OperacionManual.Properties.S_OM..cctor()	↓	-2,20	2,49	0,30
OperacionManual.Program.Main()	↑	48,94	60,40	109,34
OperacionManual.form1.tsb_stop_Click(object,class System.EventArgs)	↓	-9,13	13,17	4,04
OperacionManual.form1.tsb_conectar_Click(object,class System.EventArgs)	↓	-1,68	2,77	1,09
OperacionManual.form1.timerTestGrp_Tick(object,class System.EventArgs)	↑	2,55	0,00	2,55
OperacionManual.form1.timerSOC_Tick(object,class System.EventArgs)	↓	-87,84	366,95	279,11
OperacionManual.form1.form1_FormClosing(object,class System.Windows.Forms.FormClosingEventArgs)	↓	-2,90	13,98	11,07
OperacionManual.form1..ctor()	↑	97,76	178,78	276,54
OperacionManual.Backend.CrearGruposMonitoreo()	↓	-2,02	10,21	8,20

Figura 4.25: Caso M/SI v/s Caso M/I - Elapsed Exclusive values, Funciones - archivo *"Operación Manual.exe"*

Comparison Column		Delta	Baseline Value	Comparison Value
OperacionManual.form1.CrearGrupoTran()	↑	3	1	4
OperacionManual.form1.CerrarGrupoTran()	↑	3	1	4
OperacionManual.form1.buttonredt19pm810()	↑	6	1	7
OperacionManual.form1.buttonredt19pm750()	↑	6	1	7
OperacionManual.form1.buttonredt15pm810()	↑	6	1	7
OperacionManual.form1.buttonredt15pm750()	↑	6	1	7
OperacionManual.form1.buttonModoAutoConfigOK_Click(object,class System.EventArgs)	↑	1	0	1
OperacionManual.form1.buttonimaestro()	↑	6	1	7
OperacionManual.form1.buttonAutoFijo_Click(object,class System.EventArgs)	↑	1	0	1
OperacionManual.form1.button_TrackingPV_ManualON_Click(object,class System.EventArgs)	↑	4	0	4
OperacionManual.form1.button_TrackingPV_ManualOFF_Click(object,class System.EventArgs)	↑	4	0	4
OperacionManual.form1.button_TrackingPV_AutoOFF_Click(object,class System.EventArgs)	↑	1	0	1
OperacionManual.form1.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	↑	331	21	352
OperacionManual.Backend.Substring(string,int32)	↓	-88	4.419	4.331
OperacionManual.Backend.CompleteHandler2(object,class OPCDA.NET.RefreshEventArguments)	↑	10	4	14
OperacionManual.Backend.CompleteHandler1(object,class OPCDA.NET.RefreshEventArguments)	↑	44	2	46
OperacionManual.Backend.agregarEvento(object,class System.Windows.Forms.ListBox,bool)	↑	331	33	364

Figura 4.26: Caso M/SI v/s Caso M/I - Número de Llamadas Funciones - archivo *"Operación Manual.exe"*

Comparison Column		Delta	Baseline Value	Comparison Value
Planificacion.dll	↓	-24	1.106	1.082
OperacionManual.exe	↑	3.049	13.357	16.406
Microsoft.VisualBasic.PowerPacks.Vs.dll	↑	66	345	411

Figura 4.27: Caso M/SI v/s Caso M/I - Número de Llamadas Módulos - archivo *"Operación Manual.exe"*

4.2. Discusión

Tras haber presentado el análisis del código del programa y su respectivo diagrama UML, se propuso y se realizó una modularización orientada a separar el archivo *"form1"* en un Backend y un Frontend. Una vez realizada la modularización se propusieron 4 casos: el caso base sin modularizar, con interacción y sin interacción y el caso modularizado con interacción y sin interacción. Para estudiar estos casos se utilizó la herramienta *"Performance Wizard"* para elaborar un informe completo de los elementos analizados. Esta recolecta estadísticas de la aplicación que ayudan a descubrir problemas de rendimiento en el código o saber qué aplicaciones son las que consumen más CPU por ejemplo.

Particularmente en este caso, se consideró para el análisis solo los reportes arrojados por el programa Visual Studio, de manera de notar como cambia el rendimiento entre cada caso y como las funciones se dividen el trabajo entre ellas.

Luego de realizada la modularización se puede apreciar que aparece una nueva función *"System.Windows.Forms.Application. Run(class.System.Windows.Forms.Form)"*, la cual se divide la carga de trabajo con la otra función del mismo tipo preexistente. Esto hace notar que donde antes habia un solo proceso corriendo, el *"form1"*, ahora esa tarea se dividió en dos, por tanto, la modularización no solo es efectiva en dividir el archivo sino también la forma en como se procesan las tareas.

Respecto al trabajo anterior del SCADA, se puede notar que no hubo un esquema de trabajo riguroso ni un código comentado de manera ordenada; e incluso aunque habían partes comentadas, no existía un documento de respaldo como un léeme que explicara de qué se trataba, cómo se corría y cómo operaba el programa. Tampoco se encuentran en línea las versiones para tomarlas y hacer pruebas con ellas. Es por esta razón que la complejidad del trabajo estuvo en primero, comprender que hacía el programa, luego detectar el espacio de posible mayor problema y proponer una solución allí. Esto trajo como resultado final y aporte principal, tener un código mejor comentado y ahora con el Backend parcialmente separado del Frontend, lo que permite en un futuro poder separar el trabajo y las modificaciones en dos frentes.

Referente a la modularización realizada queda pendiente mover funciones referentes al *timer* y a los otros elementos del sistema. Esta modularización debe seguir la misma lógica de como se realizó hasta ahora, mover funciones y variables y cuando sean requeridas por otra función del "otro lado" (de Backend a Frontend y viceversa), referirla como *"miBackend.variable_Backend "* si se requiere en el Frontend o bien como *"frontEnd.variable_Frontend"* si se requiere en el Backend, esto va a depender de donde se declare la variable

En cuanto al funcionamiento mismo del SCADA, este no se corrió usando el EMS y por tanto no consideró los datos históricos para el uso de la microrred. En un futuro se vislumbra como solución poder empaquetar un EMS que se pueda acoplar a un SCADA y pueda recibir diferentes consignas que le permitan operar como gestor de la microrred, al mismo tiempo que se pueda empaquetar un programa SCADA de manera de funcionar junto al EMS.

Para realizar la experiencia de prueba se utilizaron los software *Microsoft Visual Studio 2010* para la programación del código, además de *MatrikonOPC Explorer* y *MatrikonOPC Server for Simulation and Testing* para la implementación y prueba del Servidor OPC. Además para comprobar el funcionamiento del programa, se diseñó una prueba replicable donde se explica paso a paso lo que se prueba en el programa. Estos datos permiten realizar nuevamente pruebas al programa y ver mejoras o fallas en su desempeño.

Finalmente, los resultados generales concuerdan con las expectativas: el programa modularizado se separa en dos procesos llamados *form* de manera que el uso de la única función *form* que consume casi el 90 % del uso de la CPU se separa en dos procesos que consumen alrededor de 40 % de la CPU cada uno por separado. Esto en términos concretos,es decir la modularización que se realizó sobre el código del programa, reduce las fallas y facilita la resolución de los problemas ya que se puede separar más claramente los sectores del software donde hay que intervenir para resolverlos. Además, dado que este programa se encuentra implementado en una localidad alejada, impacta significativamente en la reducción de los viajes a esta zona y ahorra tiempo y recursos para el Centro de Energía y quienes monitorean constantemente esta red y el software que utiliza.

Capítulo 5

Conclusión

En este trabajo de memoria se abordó principalmente la modularización del código del software SCADA implementado en la actualidad en la microrred de Huatacondo. Para ello se utilizó la metodología presentada en el capítulo 3, la cual describe los pasos a seguir desde el levantamiento de información del programa, pasando por la identificación de problemas y oportunidades de mejoras, la posterior propuesta de mejoras y su implementación. Tal como se muestra en el capítulo 4, se trabajó sobre el arhivo *"form1"* y se logró separar una parte del mismo en dos partes que operan como FrontEnd y Backend, quedando el nuevo Frontend donde antes se hallaba el *"form1"*, mientras que se creó una clase llamada *"Backend"* dentro de *'Operación Manual"* que es el *"form1"*como se aprecia en la Fig. 4.1.

Se logró conectar además ambas clases (Frontend y Backend) a través de las variables *"miBackend"* perteneciente a la clase Backend, ubicada en el Frontend y un objeto *"frontEnd"* de la clase *"form1'*, declarada en el Backend. Estos objetos sirven para conectar funciones que están declaradas en el otro lado, es decir, a las del Backend para las que existen en el Frontend y a las del Frontend para las que existen en el Backend.

Se logró el objetivo de modularizar el archivo *"form1"* en Frontend y Backend el archivo *"form1"*, definiendo claramente regiones para las Variables, el Constructor y los Métodos. En particular, se movieron a la región Métodos del Backend, las *Funciones Handler, OPC y Asíncrona*, además de las *Funciones de Gestor de Servidor y Grupos OPC*, las cuales permiten mayoritariamente la comunicación. Con ellas se reubicaron también las funciones *CompleteHandler1* y *CompleteHandler2*, encargadas de parte de la comunicación con las unidades. En el Frontend también se hicieron cambios: las anteriores funciones desaparecieron de ahí y en su lugar hay funciones "puente" que permiten llamar funciones desde el Frontend al Backend.

Se revisó el estado del arte de las arquitecturas más utilizadas a nivel mundial para abordar el problema de control del SCADA. Se mejoró el desempeño del sistema mediante la modularización del SCADA. Se propuso un esquema de modularización que fue altamente efectivo al separar las funcionalidades del *"form1"* en un Backend y un Frontend. Además se logró concretar esto como dos clases separadas estructuralmente solo unidas por variables que permitían un puente entre ambas clases. Además se probó que el desempeño mejora y no existe solo un proceso ejecutándose, sino que dos separados andando en paralelo: uno es el Backend y el otro el Frontend.

El aporte de este trabajo impacta significativamente sobre los recursos volcados en mejorar, supervisar y actualizar el programa ESUSCON que maneja la microrred de Huatacondo, debido a que la modularización del mismo permite identificar de forma más fácil las partes del código donde se producen problemas y resolverlos, además de definir las estructuras del programa como el Backend y el Frontend y esto trae consigo una reducción en el uso de recursos como tener que viajar menos a Huatacondo en caso de fallas, menor tiempo en detección de problemas con el código y menos gasto de recursos para quienes administran esta microrred.

Finalmente, al analizar los 4 casos estudiados, se puede ver un número menor de llamadas a funciones en los casos modularizados y la existencia de 2 forms que se ejecutan por separado ocupando recursos diferentes de la CPU; además las funciones presentes en el *Hot Path* son más y se dividen la carga de trabajo. Luego, en los casos con interacción, se aprecia en el reporte el mayor uso de la CPU y de la función *System.Threading.Thread.Sleep(int32)*, encargada de detener algunas tareas por cuestión de milisegundos para que otras tareas puedan ser ejecutadas, lo que hace sentido ya que en estos casos se interactuó con el programa.

5.1. Trabajo Futuro

Como trabajo futuro se deja propuesto terminar de modularizar el codigo separándolo en clases funcionales: PV, Diesel, Historizador, Baterías, EMS y Planificación como se mencionó en la sección 3.3.1. Estas clases hoy no se encuentran disponibles como tales en el archivo con el cual se trabajó.

También se plantea indagar más en las estructuras de control que se utilizan en el control de la microrred y como sus componentes se coordinan. Una propuesta puede ser migrar a una arquitectura descentralizada con la idea de que cada clase maneje componentes por separado pero que a la vez sde coordinen.

Otro trabajo futuro que se visibiliza es migrar el programa ESUSCON desde el lenguaje C# en el cual se encuentra implementado a *Python* ya que es un lenguaje más popular en la actualidad y a diferencia de C# es OpenSource.

Se plantea además realizar una mejora en el control de excepciones y probar el funcionamiento del software añadiendo el uso del EMS y también los datos históricos de la microrred de Huatacondo.

Dado finalmente que la modularización solo se realizó en el archivo *"form1"*, se propone realizar el proceso de modularización en otros archivos del mismo programa, lo que permitiría un mejor desempeño de este mismo y un orden más definido y claro de lo que es y lo que hace cada parte del código.

Glosario

- **DER:** Distributed Energy Resources
- **UML:** *Unified Modeling Language* o Lenguaje Unificado de Modelado
- **SCADA:** Sistema de Control y Adquisición de Datos
- **EMS:** Energy Management System
- **OPC:** Estándar de comunicación en el campo del control y supervisión de procesos industriales
- **Backend:** Capa de acceso y procesamiento de datos de un programa
- **Frontend:** Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios.
- **Diagrama de Clases:** Diagrama que muestra la relación entre las clases que operan y existen en un programa.
- **p.u.:** o *"por unidad"* de cualquier cantidad se define como la relación entre esta cantidad y la cantidad base y se expresa como un decimal.
- **Open Source:** Se dice de un programa o aplicación que tiene código abierto, es decir, es modificable y no privativo.

Bibliografía

[1] F. J. Lanas Montecinos, "Desarrollo y Validacion de un Modelo de Optimizacin Energtica para una Microrred," 2011.

[2] J. Kaldellis and D. Zafirakis, "The wind energy (r)evolution: A short review of a long history," *Renewable Energy*, vol. 36, pp. 1887–1901, 07 2011.

[3] E. Bianchi, ""ELEMENTOS DE ELECTROQUÍMICA: ELECTROLISIS Y ACUMULADORES REVERSIBLES". Apunte curso EL607-1 Aplicaciones Industriales de la Energía Eléctrica," Universidad de Chile, Tech. Rep., 2011.

[4] J. Oyarzabal, N. H. T. Degner, A. S. C. S. D. D. Geibel, and J. Yarza, "Advanced Architectures and Control Concepts for More Microgrids," no. January, 2007.

[5] M. Bertocco and F. Tramarin, "A system architecture for distributed monitoring and control in a smart microgrid," in *2012 IEEE Workshop on Environmental Energy and Structural Monitoring Systems (EESMS)*, Sep. 2012, pp. 24–31.

[6] F. Katiraei, R. Iravani, N. Hatziargyriou, and A. Dimeas, "Microgrids management," *IEEE Power and Energy Magazine*, vol. 6, no. 3, pp. 54–65, May 2008.

[7] J.L. Leiva O., *Diseño de Algoritmos*.

[8] M. Starke, B. Xiao, G. Liu, B. Ollis, P. Irminger, D. King, A. Herron, and Y. Xue, "Architecture and implementation of microgrid controller," in *2016 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, sep 2016, pp. 1–5.

[9] M. Saleh, Y. Esa, and A. A. Mohamed, "Communication-Based Control for DC Microgrids," *IEEE Transactions on Smart Grid*, vol. 10, no. 2, pp. 2180–2195, 2019.

[10] M. Saleh, Y. Esa, A. A. Mohamed, H. Grebel, and R. Rojas-Cessa, "Energy management algorithm for resilient controlled delivery grids," in *2017 IEEE Industry Applications Society Annual Meeting*, Oct 2017, pp. 1–8.

[11] L. Mateen, N. A. Zaffar, M. Nasir, A. Hussain, and H. A. Khan, "Solar PV-Based Scalable DC Microgrid for Rural Electrification in Developing Regions," *IEEE Transactions on Sustainable Energy*, vol. 9, no. 1, pp. 390–399, 2017.

[12] J. M. Guerrero, E. A. A. Coelho, R. Han, G. Ferrari-Trecate, J. C. Vasquez, and L. Meng,

"Containment and Consensus-Based Distributed Coordination Control to Achieve Bounded Voltage and Precise Reactive Power Sharing in Islanded AC Microgrids," *IEEE Transactions on Industry Applications*, vol. 53, no. 6, pp. 5187–5199, 2017.

[13] T. Schnelle, A. Schweer, and P. Schegner, "Control of modular microgrids by varying grid frequency," *2017 IEEE Manchester PowerTech, Powertech 2017*, 2017.

[14] A. C. Luna, E. Rodriguez, J. C. Vasquez, L. Meng, A. Luna, and T. Dragicevic, "Flexible System Integration and Advanced Hierarchical Control Architectures in the Microgrid Research Laboratory of Aalborg University Savaghebi , Juan Vasquez , Josep Guerrero Moisès Graells Fabio Andrade," *Ieee Transactions on Industry Applications*, vol. 52, no. 2, pp. 1736–1749, 2016.

[15] Q. Xu, P. Wang, Y. Zhang, C. Wen, and J. Xiao, "A decentralized control strategy for economic operation of autonomous AC microgrids," *IECON Proceedings (Industrial Electronics Conference)*, vol. 32, no. 4, pp. 4020–4024, 2016.

[16] F. Dörfler, J. Simpson-Porco, F. Bullo, E. Mojica-Nava, C. A. Macana, N. Quijano, C. Barreto, N. Quijano, J. Giraldo, E. Mojica-Nava, and N. Quijano, "Breaking the Hierarchy: Distributed Control & Economic Optimality in Microgrids," *IEEE Transactions on Control of Network Systems*, vol. 44, no. 3, pp. 1–13, 2014. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6663743{%}5Cnhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7164313{%}5Cnhttp://arxiv.org/abs/1401.1767

[17] T. Morstyn, A. V. Savkin, B. Hredzak, and H. D. Tuan, "Scalable Energy Management for Low Voltage Microgrids Using Multi-Agent Storage System Aggregation," *IEEE Transactions on Power Systems*, vol. 33, no. 2, pp. 1614–1623, 2018.

[18] R. Ernesto, P. Behnke, A. Omar, M. Ramos, L. Santiago, and V. Díaz, "Metodología Básica para la Definición de la Ubicación Óptima de un Generador Virtual," Ph.D. dissertation, 2009.

[19] I. Erbetta Mattig, "Predicción de la Potencia para la Operación de Parques Eólicos," Ph.D. dissertation, 2010.

[20] A. Tuckey, S. Zabihi, and S. Round, "Decentralized control of a microgrid," in *2017 19th European Conference on Power Electronics and Applications (EPE'17 ECCE Europe)*, sep 2017, pp. P.1–P.10.

[21] F. Z. Harmouch, N. Krami, N. Hmina, and E. H. Margoum, "A multiagent based hierarchical control for Microgrid cluster stability enhancement," in *2016 International Renewable and Sustainable Energy Conference (IRSEC)*, nov 2016, pp. 1034–1039.

[22] A. Bani-Ahmed, A. Nasiri, and H. Hosseini, "Design and development of a true decentralized control architecture for microgrid," in *2016 IEEE Energy Conversion Congress and Exposition (ECCE)*, sep 2016, pp. 1–5.

[23] A. Solanki, A. Nasiri, V. Bhavaraju, T. Abdullah, and D. Yu, "A new control method for

microgrid power management," in *2013 International Conference on Renewable Energy Research and Applications (ICRERA)*, Oct 2013, pp. 1212–1216.

Apéndice A

Códigos

A continuación se muestra el código donde se hizo hincapié en realizar cambios en el SCADA.

A.1. Constructor del *"form1"*

El *"form1"* se trata de un constructor que sirve de inicialización al programa. Para ello se utiliza el comando *Application.Run()* en C# dentro de las declaraciones de la clase.

Para muchas aplicciones debe existir un punto de partida. .NET refiere a la clase del programa con la cual será creada automáticamente para aplicaciones de"forms".

Figura A.1: Inicialización del "form1"

diag de dependencia de clases.dgml* | Program.cs

OperacionManual.Program | Main()

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace OperacionManual
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new form1());
        }
    }
}
```

Apéndice B

Rutinas

A continuación se adjunta la URL del repositorio donde se trabajó con los archivos del SCADA si se desea mayor consulta: https://github.com/rugartemunoz/SCADAhuatacondo.git