**ARQUITECTURA DE SOFTWARE**

**Regulador de Turbina**

1. Objetivo **3**

2. Definiciones **3**

3. Descripción general de la arquitectura **3**

4. Regulador **5**

4.1. Control y Regulation Process **6**

4.2. IO Image **8**

4.3. IO Field Driver **9**

4.4. IO Slave Driver **10**

4.5. Alarm Manager **11**

5. HMI Local **12**

5.1. Regulator Model **13**

5.2. Supervisory Control Module **14**

5.3. Alarm Viewer **15**

5.4. RealTime Measurer **15**

5.5. RealTime Tendency Viewer **15**

5.6. Master Driver **16**

6. Configuration module **17**

1. Objetivo

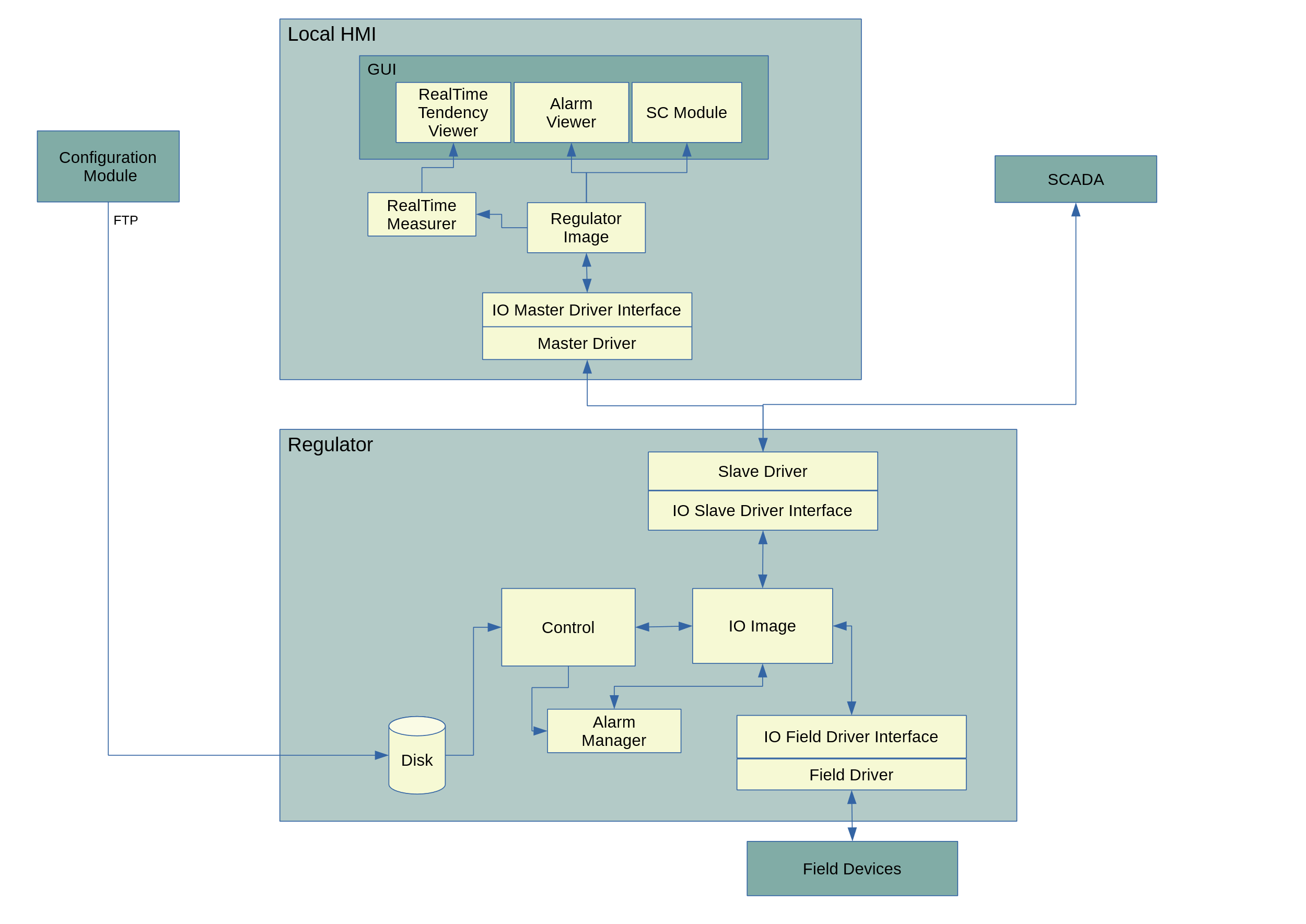
En este documento se describe el diseño y la implementación del software empleado en el regulador de turbina.

1. Definiciones

|  |  |
| --- | --- |
| **TERMINOS** | **DEFINICIONES** |
| *HMI* | *Human Machine Interface* |
| *SCADA* | *Supervisory Control And Data Acquisition* |

1. Descripción general de la arquitectura

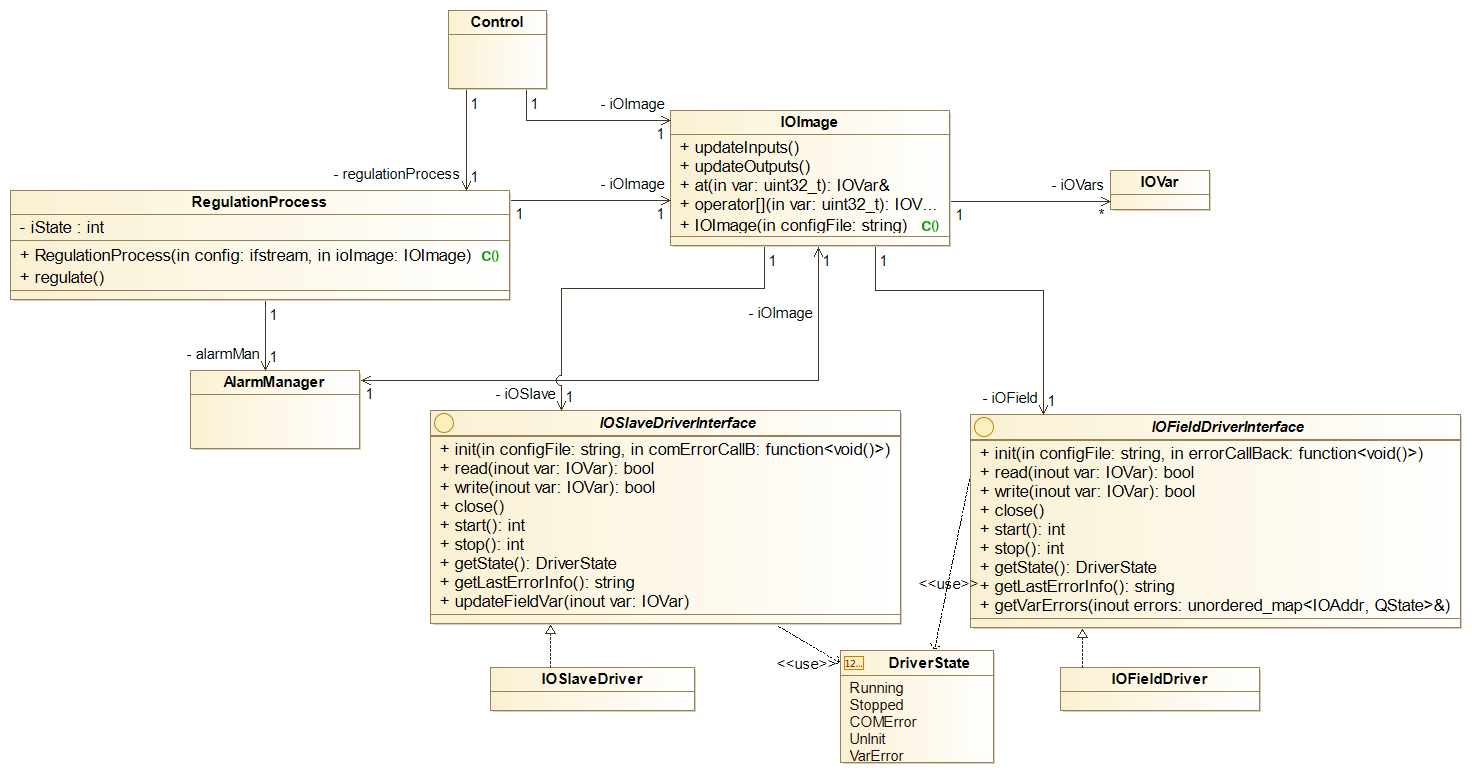
En el regulador de turbina, además de proporcionar la funcionalidad de regulación, se provee de la capacidad de comunicación con los elementos del campo, así como con un maestro el cual pueda comandar y consultar al regulador. También se dota al mismo de la capacidad de configuración de sus parámetros de funcionamiento y muestra de medidas a tiempo real y alarmas.

Para esta funcionalidad se presenta la siguiente solución, dividida en módulos independientes, algunos de los cuales podrían estar en una maquina diferente a la del regulador:

Los módulos presentados se describen como sigue:

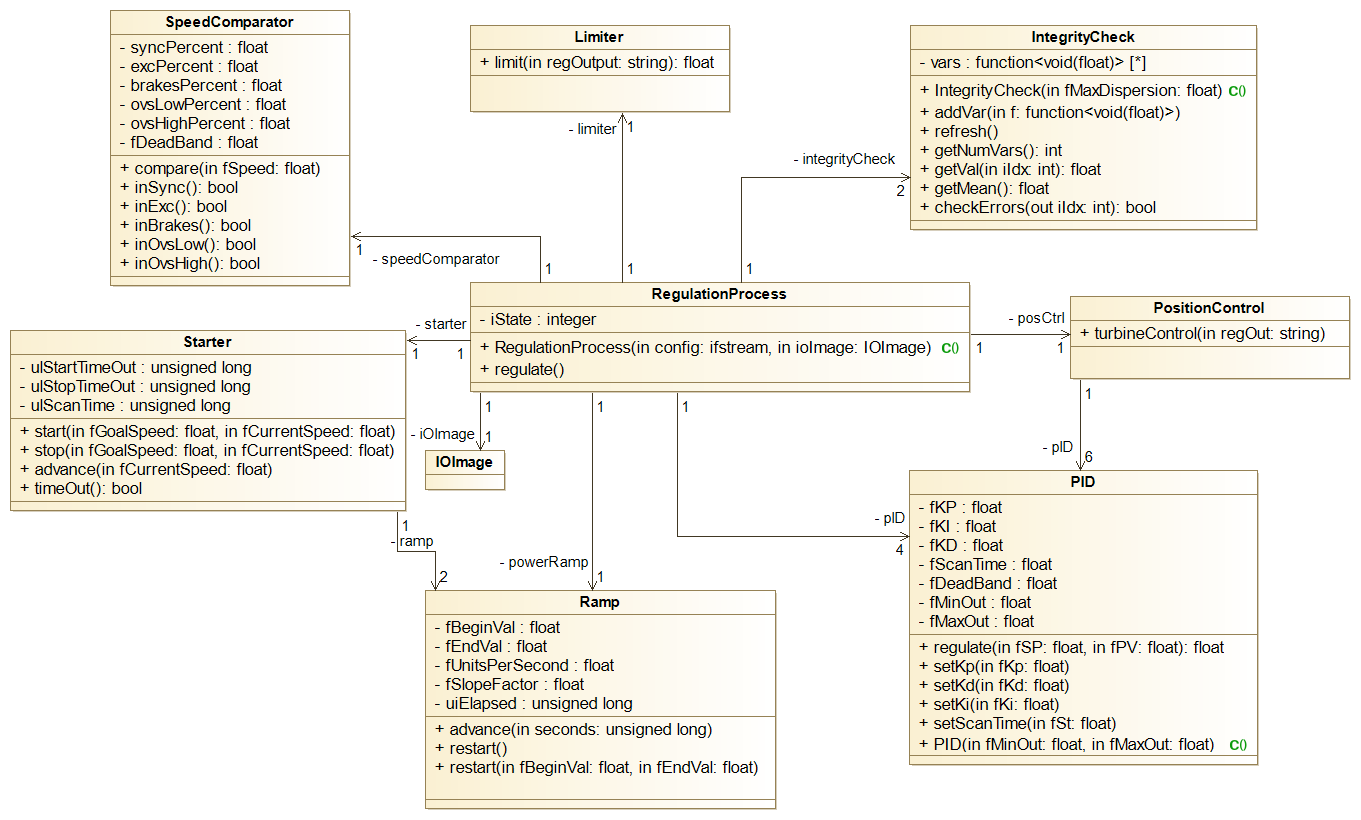
* **Regulador:**
* [**Control**](#Control): Es el módulo principal del regulador. Contiene el bucle principal del programa, en el que se actualizan las entradas y salidas a través de la IO Image y se generan salidas de control.
* [**IO Image**](#IOImage): Es la representación del estado del regulador. Este estado se actualiza antes y después de la regulación, leyendo las entradas del campo y comandos del maestro y escribiendo al final sus cambios en los mismos.
* **IO Field Driver Interface**: Representa la interfaz del driver de comunicación con los elementos del campo.
* [**Field Driver**](#fieldDriver): Es el driver que implementa la anterior interfaz de comunicación.
* **IO Slave Driver Interface**: Representa la interfaz del driver de comunicación con el maestro del regulador.
* [**Slave Driver**](#slavedriver): Es el driver que implementa la interfaz de comunicación con el maestro del regulador.
* [**Alarm Manager**](#alarmmanager): Modulo que encapsula la generación de alarmas. Permite notificar estados de error codificando cada uno la activación de las salidas asociadas.
* **HMI Local**:
* **Regulator Image:** Representación abstracta del regulador. Ofrece una interfaz hacia el resto de la aplicación para interactuar con el regulador, encapsulando su acceso.
* **Supervisory Control Module:** Modulo encargado de mostrar en la GUI el estado de las variables del regulador y de aplicar comandos sobre este.
* **RealTime Measurer:** Modulo que mantiene y registra una serie de muestras sobre distintas variables del proceso. Para ello utiliza múltiples colas FIFO que actúan como buffers.
* **RealTime Tendency Viewer:** Vista en forma de graficas de las medidas tomadas del proceso.
* **Alarm Viewer:** Vista y gestión de las alarmas producidas en el regulador.
* **IO Master Driver Interface:** Interfaz que especifica un driver de comunicación con el regulador esclavo.
* **Master Driver:** Driver de comunicación que implementa la anterior interfaz.
* **Configuration Module**: Es el programa que genera el archivo de configuración del regulador. Se encuentra en una máquina distinta.

1. Regulador



* 1. Control y Regulation Process

El objetivo del regulador es controlar la turbina de una forma adecuada y segura, respondiendo a los cambios en el entorno y a las acciones del operador. El módulo de control contiene el bucle principal del proceso, en el que se actualizan entradas y salidas del sistema y se generan salidas de control. Esto último es encapsulado por el Regulation Process (junto con otros módulos) y consiste básicamente en la generación, para cierto estado del regulador, de una salida de apertura de los elementos de control de la turbina.

Para ello se propone la siguiente estructura de clases:

* **RegulationProcess**: Clase principal que encapsula todo el proceso de regulación. Incorpora un autómata de estados sobre el tipo de regulación en curso, ya sea en vacío, apertura, etc.
* **Limiter**: Limitador de salida en caso de disparo, o como protección para el arranque o regulación.
* **PositionControl**: Clase que encapsula el control de posición de los elementos de cada tipo de turbina. Recibe la salida del limitador.
* **SpeedComparator**: Clase que muestra si ciertas condiciones sobre la velocidad se cumplen. Estas son representadas mediante porcentajes de velocidad, que una vez hayan sobrepasado (por encima o por debajo) mostraran la condición activa en el comparador, i.e. velocidad de sincronización alcanzada (90% de velocidad alcanzado). Incorporará una banda muerta para evitar la histéresis en la activación de las condiciones.
* **IntegrityCheck**: Encapsula la comprobación de la validez, de las medidas de varios sensores de una misma variable (i.e. sensores redundantes 2 de 3). Estas variables son capturadas por el objeto como punteros a función y actualizadas en cada ciclo de regulación mediante una acción externa.
* **Ramp**: Clase que proporciona una salida en forma de rampa. Como parámetros se fijan las unidades de tiempo en las que se trabaja, el ratio de cambio por unidad de tiempo y un factor para ajustar la pendiente de la rampa. Se inicia con un valor inicial y final y proporciona una salida en función del tiempo transcurrido.
* **Starter**: Modulo para arrancar y parar el regulador. Ofrece dos comandos para iniciar estas acciones, en los que se especifica velocidad destino y actual (en porcentaje), y genera una salida en rampa para cada caso. Incorpora un autómata de estados para llevar cuenta de estos procesos. Para la generación de la salida se llama a la función advance con la velocidad actual. Tiene una función que indica si se ha producido TimeOut en alguno de los dos procesos y

una para indicar que se ha completado el proceso de arranque.

La configuración de los distintos elementos de control será descrita en la [sección 6](#ConfigModule).

* 1. IO Image

Es la representación del estado del regulador, comprende todas las variables de entrada y salida. Para ello utiliza los drivers de campo y esclavo para recoger los cambios y actualizarlos en sus variables.

Implementa métodos para la actualización de entradas y salidas y para consultar los valores de las variables. Para ello utiliza la siguiente estructura para su codificación:

|  |  |
| --- | --- |
| **IOVar** | |
| **uint32\_t** | muiVarID |
| **int64\_t** | mulTimeS |
| **float** | mfCurrentVal |
| **float** | mfForcedVal |
| **float** | mfEguMin |
| **float** | mfEguMax |
| **float** | mfHWMin |
| **float** | mfHWMax |
| **IOAddr** | mtAddr |
| **double** | mdPulseDur |
| **int64\_t** | miPulseTimeS |
| **bool** | mbForcedVal |

|  |  |
| --- | --- |
| **IOAddr** | |
| **uint8\_t** | uiHeader |
| **uint8\_t** | uiModule |
| **uint8\_t** | uiChannel |
| **uint8\_t** | uiBitPreccission |

El direccionamiento de las variables en los drivers viene dado por la cabecera, el módulo, el canal y el número de bits que direcciona.

Los rangos de HW son utilizados por el driver de campo para linealizar el valor de medición (i.e. medición de 0 a 32767 linealizado de 0 a 16000).

Los rangos Egu son usados para linealizar la variable en porcentaje en la regulación.

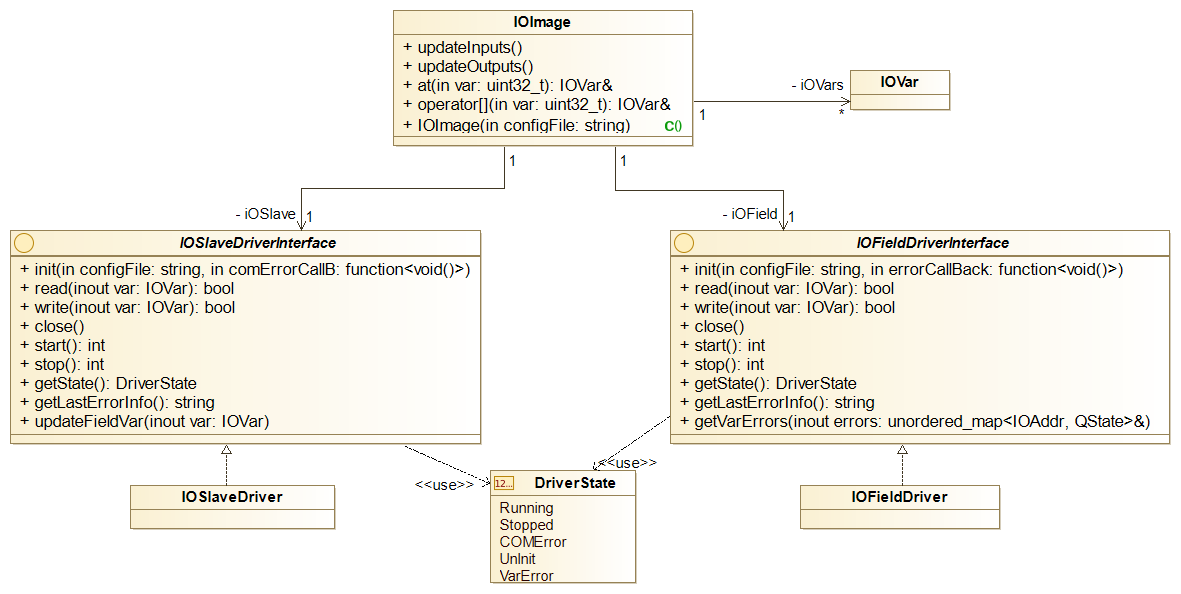
La duración de pulso y su TimeStamp recogen el tiempo de activación del ultimo pulso de la señal digital y cuando fue recogido.

El campo mbForcedVal permite indicar que la variable esta forzada y su valor actual pasa de ser mfCurrentVal a mfForcedVal. El forzado de variables está pensado para simular los valores de entrada y salida que maneja el driver de campo. Para ello el driver esclavo tendrá direccionables estas variables y desde el maestro será posible escribir en el flag de forzado y su valor. Desde otras clases que consulten el valor de la variable se devolverá el valor forzado o el actual en cada caso. Para más información consultar la [sección 4.4](#slavedriver).

La clase recibe en configuración los datos de las variables usadas en el regulador.

La estructura y datos del archivo de configuración esta descrita en la [sección 6](#ConfigModule).

Las variables estarán separadas en 4 estructuras de datos: entradas y salidas de campo, y entradas y salidas de esclavo.



* 1. IO Field Driver

Este driver tiene como objetivo controlar las E/S del campo, proporcionando una interfaz para que el regulador consulte y escriba en estas.

El driver aparte de leer y escribir en los dispositivos tendrá que medir la duración de los pulsos de determinadas variables digitales.

Se implementara el driver con el protocolo Ethercat. El direccionamiento de una variable en Ethercat se define mediante un esclavo en la red y un canal de entrada o salida dentro del mismo. Junto con la precisión se puede localizar la variable en el mapa de E/S.

En la configuración recibirá la dirección y datos de las variables a manejar. Estas variables, al representar módulos con funcionalidad diferente, se trataran como objetos diferentes representantes de la categoría de módulos de la variable. Estos objetos además de codificar las operaciones de lectura y escritura recibirán llamadas del driver a lo largo del ciclo de comunicación para actualizar sus medidas, condiciones, etc.

Estos representantes serán:

* DigitalDev: Representa una variable digital. En su actualización puede medir la duración de los los pulsos por alza o baja.
* AnalogDev: Representa una variable analógica. En su actualización revisara sus condiciones de error y notificara al driver. Como parámetros recibe la dirección de la palabra de estado (hasta 32 bit) para comprobar errores y la codificación binaria de los valores que recibe (Unsigned, Signed y HalfSigned de 0 a 32767). Puede tratar con dispositivos de hasta 32 bit.
* PulseDev: Representa un contador de pulsos. Recibe como parámetro la dirección de la palabra de control y el bit de set del contador, para habilitar escrituras. Puede tratar con contadores sin signo de hasta 32 bits.

En la inicialización también recibe una lista para la configuración de los esclavos, que contendrá en cada fila el esclavo objetivo, el índice y el subíndice de configuración en el esclavo Ethercat y el valor a escribir.

La estructura y datos del archivo de configuración esta descrita en la [sección 6](#ConfigModule).

El driver gestionara 3 tipos de errores:

* Error de comunicación: Se pierde la comunicación con los esclavos Ethercat. El driver notifica a través de un callback para su consulta. Todas las variables leídas son marcadas con un estado de calidad de ComError.
* Error de esclavo: No se recibe respuesta de un esclavo en el ciclo de comunicación. Las direcciones asociadas al modulo son introducidas en una lista de variables con error y se notifica mediante un callback para su consulta.
* Error de variable o canal: La variable a sufrido un error de medida (i.e. UnderRange, OverRange) La variable avisa al driver e introduce su dirección en la lista de variables erróneas. El driver notifica como antes.

La notificación con los callback se hará una vez al detectar el error. Hasta que no deje de estar en estado de error y vuelva otra vez permanecerá sin cambios.

El driver contara con funciones de control como inicio y parada del proceso, además de una función DriverState que retornara el estado del driver, véase Running, Stopped, COMError (Error por perdida de conexión), UnInit y VarError (Error en dispositivos de driver).

* 1. IO Slave Driver

Es el driver encargado de manejar las peticiones de consulta y comando, realizadas por uno o varios maestros que supervisan y controlan la regulación.

El driver esclavo y maestro utilizan un direccionamiento propio, las variables se numeran de 1 a N. Internamente la comunicación de estos esta implementada en ModBus. El driver esclavo tiene los siguientes campos repartidos en sus tablas para cada variable:

|  |  |  |
| --- | --- | --- |
| **Variable 1** | |  |
| **Dirección** | **Campo** | **Reparto en tablas MB** |
| **0001** | Bit de forzado | 1 Coil |
| **1001-1002** | Estado de calidad | 2 Discrete Inputs |
| **3001-3004** | TimeStamp | 4 Input Registers |
| **4001-4002** | Valor actual | 2 Holding Registers |
| **4003-4004** | Valor forzado | 2 Holding Registes |

Los campos de las sucesivas variables se asignan de marera consecutiva, por lo que a partir del numero de variable se puede encontrar sus campos en cada tabla por desplazamientos.

El estado de calidad ocupa dos bits, y el más significativo es escrito en el primer bit.

El valor actual y el forzado son dos variables float, escritas en Big Endian.

El time stamping son 8 bytes empezando por el más significativo.

Las operación de lectura del driver lee el valor actual de la variable y genera un estado de calidad y time stamping. La operación de escritura solamente escribe su valor en la tabla.

El driver incluye la capacidad de forzar el valor de variables de campo. Para ello en configuración recibe una lista con sus identificadores y direcciones en el driver (1 a N). El driver incorpora una función de actualización para estas variables, la cual escribirá el valor actual, el time stamping y el estado de calidad de la variable en la tablas, para su consulta por los maestros, y copiara en la variable el valor de forzado y su bit.

La estructura y datos del archivo de configuración esta descrita en la [sección 6](#ConfigModule).

El driver solamente entra en estado de error si no puede establecer un servidor a los maestros.

El driver implementara los mismos métodos de control que el Field Driver.

* 1. Alarm Manager

El gestor de alarmas es el módulo encargado de proporcionar una interfaz de gestión de estados de error para las clases del regulador. Proporciona una tabla con los distintos estados de error a generar y permite asociar a un estado de error la activación de distintas salidas del campo y/o esclavo. (i.e. Disparo activa la salida de campo de disparo y la de alarma mayor, además de la salida de esclavo de disparo).

Además, gestionara la codificación de las alarmas en una máscara de bits.

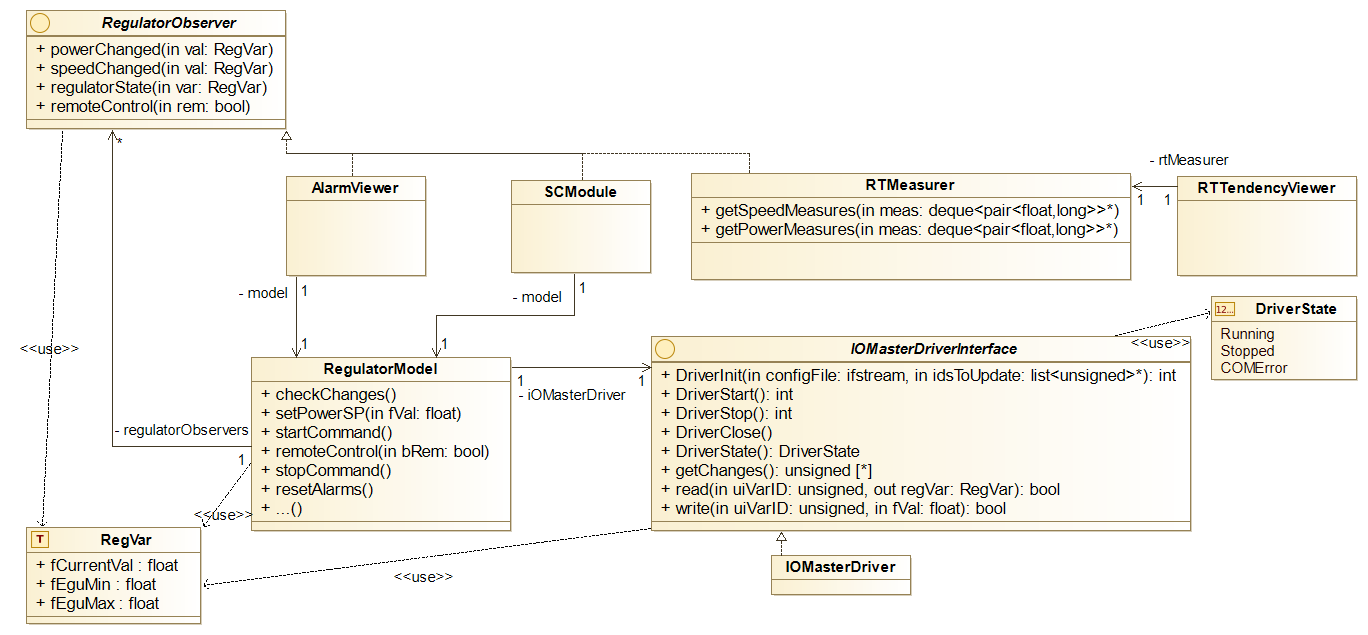
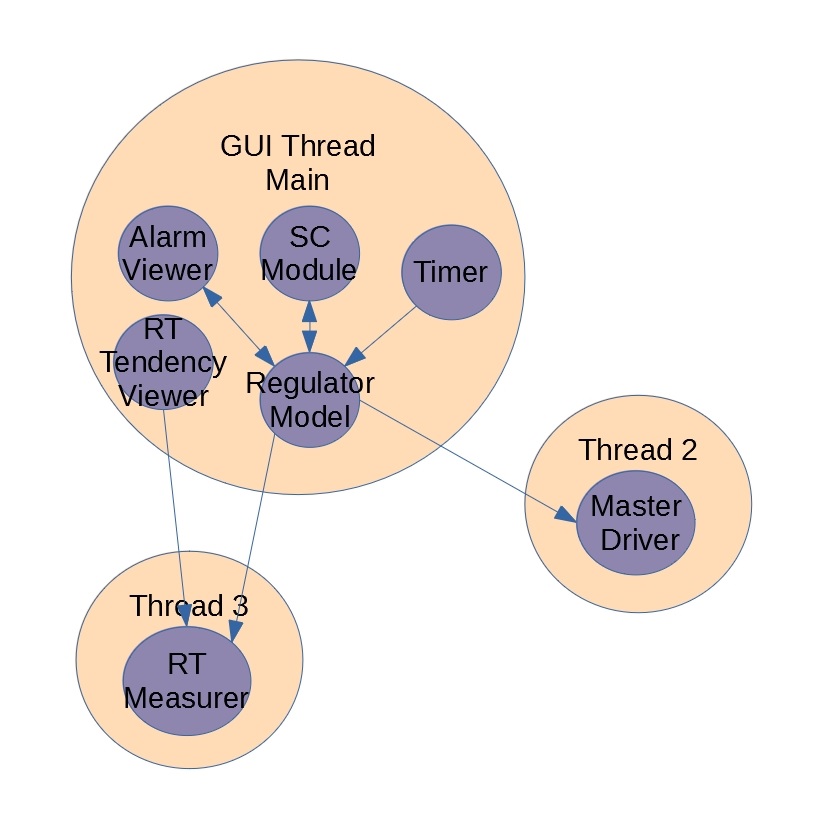
Los estados de alarma son los siguientes, cada uno de los cuales pueden activar distintas salidas del regulador:

* + **Alarmas mayores**
    - Error de conexión con dispositivo E/S
    - Rotura de conexión con módulo de campo
    - Disparo de emergencia
    - Sobrevelocidad en nivel alto
    - TimeOut en arranque
    - TimeOut en parada
    - TimeOut en descarga
    - Error de medida de velocidad
    - Error de medida de potencia
    - Fallo en configuración
    - Regulación inestable
  + **Alarmas menores**
    - Sobrevelocidad en nivel bajo
    - Error de medida en variable de velocidad 1
    - Error de medida en variable de velocidad 2
    - Error de medida en variable de velocidad 3

1. HMI Local

El HMI Local permite la supervisión y control del proceso de regulación, a través de una interfaz cercana al dispositivo. Para mantener cierta flexibilidad y compatibilidad en cuanto a su implementación (será en Linux, pero con opción de uso en Windows, incluso en una maquina externa) se utilizará el toolkit Qt para la GUI.

El programa se dividirá en las clases ilustradas por el siguiente diagrama:



* 1. Regulator Model

El modelo del regulador es una representación abstracta del regulador físico. El objetivo del modelo es mantenerse actualizado con respecto a su sistema físico y proporcionar una interfaz sencilla a los demás módulos que requieran conocer el estado de este.

Para la actualización de las GUI o módulos que necesiten conocer su estado, se plantea una implementación similar al patrón de diseño observador. Los módulos implementan una interfaz llamada RegulatorObserver, la cual tiene métodos que representan los cambios en el regulador, i.e. cambio en la velocidad de la turbina, cambio en alarmas del regulador, etc. Estos observadores se registran en el modelo, y una vez éste detecte cambios, llamara para cada observador a las funciones representantes del cambio. En el caso de implementación en Qt esto se puede hacer con signals y slots asíncronos.

Para producir cambios en el regulador, las clases interesadas tendrán una referencia al modelo y llamaran a las funciones correspondientes relacionadas con la variable de control del regulador, i.e. arranque, parada, etc.

Para leer la configuración del regulador (número de medidas de velocidad, tipo de turbina, etc.) se podrá consultar a modelo de la misma forma que los comandos.

Las variables escritas y leídas más destacables son las siguientes:

|  |  |
| --- | --- |
| Control auto/manual | Escritura y lectura |
| Control remoto/local | Escritura y lectura |
| Estado regulación (arranque, en vacío, .) | Lectura |
| SP de potencia | Escritura y lectura |
| Señales del comparador de velocidad | Lectura |
| Registro de alarmas | Lectura |
| Medidas potencia | Lectura |
| Medidas velocidad | Lectura |
| Estado de interruptor de línea y generador | Lectura |
| Posición de inyectores | Escritura y lectura |
| Posición de alabes | Escritura y lectura |
| Posición de distribuidor | Escritura y lectura |
| Apertura de deflectores | Escritura y lectura |
| Cambio de modo de regulación (apertura, potencia) | Escritura y lectura |
| Límite de salida en regulación de potencia | Escritura y lectura |
| Arranque | Escritura |
| Parada | Escritura |
| Descarga | Escritura |
| Reposición de alarmas | Escritura |
| Trip | Escritura |

En cuanto a su gestión interna, el módulo utilizara los mismos identificadores que el regulador para referirse a las variables de cara al driver. Este conocerá por su configuración las direcciones e información de las variables.

Para la actualización del modelo se consultaran al driver los identificadores cuyos valores hayan cambiado, de forma periódica, y aplicara una operación de lectura para cada uno de ellos. En caso de pérdida de conexión o inicialización del modelo se leerán todos directamente. Los identificadores los cuales haya que leer, para mantener actualizado el modelo, estarán en una lista estática que se pasara al driver en su inicialización, para que este conozca sobre que direcciones hacer las peticiones de actualización.

El modelo será llamado por un Timer del thread cada cierto intervalo de actualización.

* 1. Supervisory Control Module

Ofrece una vista y control de las variables del proceso de regulación. Implementa la interfaz RegulatorObserver, por lo que la vista se actualiza a través de llamadas del modelo. Para el control, el módulo tiene una referencia del modelo y llama a sus funciones de comando.

Las variables supervisadas serán las siguientes:

* Control en auto/manual
* Control remoto/local
* Etapas del regulador (véase parando, arrancando, regulación en vacío, regulación en isla, regulación de potencia, regulación de apertura)
* SP de potencia
* Señales del comparador de velocidad
* Medidas de potencia
* Medidas de velocidad
* Posición de inyectores
* Posición de alabes
* Posición de Distribuidor
* Apertura de deflectores
* Límite de salida en potencia

Las variables controladas serán:

* Control en auto/manual
* Control remoto/local
* Posición de inyectores (modo mantenimiento/manual)
* Posición de alabes (modo mantenimiento/manual)
* Posición de Distribuidor (modo mantenimiento/manual)
* Apertura de deflectores (modo mantenimiento/manual)
* Límite de salida en potencia
* Cambio de modo de regulación (potencia a apertura)
* Pulsos positivos y negativos en sincronización manual
* Apertura de la turbina (SP regulación de apertura)
* Arranque
* Parada
* Trip
  1. Alarm Viewer

Ofrece una interfaz con las vistas de las ultimas alarmas producidas en registro y de las alarmas activas en ese instante en el regulador. Como comando incorpora la reposición de las alarmas.

La representación de las alarmas constara de su identificador en el regulador, su descripción y el TimeStamping de su registro.

Implementara la interfaz RegulatorObserver para la actualización de las alarmas y una referencia al modelo para el comando de reposición de estas.

* 1. RealTime Measurer

El módulo es el encargado de tomar muestras periódicas de las variables del regulador. Por simplicidad solo se podrá medir un subconjunto de las variables del modelo, como las medidas de potencia, velocidad, apertura de elementos de control, etc. Mantiene una imagen de cada una de estas variables, las cuales son actualizadas por el modelo a través de la interfaz de observador.

El thread propio del objeto se encargara de tomar medidas de la imagen interna de estas variables cada cierto intervalo y guardarlas en una cola para que otras clases procesen las medidas. Los intervalos de medida y el número de medidas por cada variable se podrán configurar a través de la vista del módulo RealTime Tendency Viewer y opcionalmente por archivo.

* 1. RealTime Tendency Viewer

Proporciona una vista de las muestras tomadas por el RealTime Measurer en forma de graficas. La interfaz permitirá elegir que variable mostrar y en función de los parámetros de muestreo, se mostrara una gráfica variable/tiempo ajustada a cada tipo.

Estos parámetros de muestreo serán el intervalo de medida y el número de medidas totales, los cuales serán transmitidos al RealTime Measurer para su configuración.

En el momento de pintado de la gráfica se solicitara la cola de muestras de la variable al RealTime Measurer.

* 1. Master Driver

Implementa el maestro de comunicación con el regulador. El protocolo utilizado será ModBus/TCP.

El driver en su inicialización recibe la lista de variables Modbus que utiliza el regulador, mediante la cual creara su mapa de memoria, y también recibe una lista del modelo, con los identificadores seleccionados para las lecturas periódicas del maestro.

El mapa de memoria contendrá las variables codificadas en la siguiente estructura:

|  |  |
| --- | --- |
| **ModbusVar** | |
| **unsigned** | uiVarID |
| **float** | fCurrentVal |
| **VarAddr** | address |
| **float** | fEguMin |
| **float** | fEguMax |

|  |  |
| --- | --- |
| **VarAddr** | |
| **unsigned** | uiHeader |
| **unsigned** | uiModule |
| **unsigned** | uiChannel |
| **unsigned** | uiBitPreccission |

El modelo solicitara escrituras especificando un ID de la variable y el valor, a lo cual el driver guardara la pareja en una cola de escrituras pendientes, para que se procese en el siguiente ciclo.

En cuanto a las lecturas el modelo proporcionara una estructura de salida, a rellenar con los datos de la variable solicitada. Esta estructura es la siguiente:

|  |  |
| --- | --- |
| **RegVar** | |
| **float** | fCurrentVal |
| **float** | fEguMin |
| **float** | fEguMax |

Aparte de estos métodos implementará los de control del driver, especificados en el Field Driver del regulador.

1. Configuration module

El módulo de configuración es un programa que permite crear un archivo de configuración para el regulador.

La parte de la configuración orientada al control intentara tener su formato dividido en clases, se indicara la clase a configurar (i.e. RegulationProcess) y a continuación sus parámetros y la posible configuración de más clases internas. Con esto se dividen las dependencias de la configuración en el módulo que la necesita, solo conociendo este cómo interpretar su sección del archivo.

El programa de configuración dará además la posibilidad de configurar la disposición de las salidas en el módulo IO, su precisión analógica, la medida en tiempo de pulsos, etc. Esto es posible ya que cada variable de estado del programa tiene un identificador, y en el caso de señales de campo se puede cambiar la dirección de estas en el driver, sin tocar nada más.

El archivo podrá ser enviado al regulador siguiendo el protocolo FTP o mediante una conexión local con él regulador.

El regulador al iniciar intentara cargar el archivo de configuración. En caso de fallo se producirá un estado de disparo en el regulador y esperara a recibir un comando de reintento para iniciar de nuevo. Mientras estará esperando un nuevo archivo por FTP o por copia local.

Ciertos parámetros de la configuración se podrán cambiar a lo largo del funcionamiento normal del regulador, a través de direcciones en el esclavo de comunicación. Entre estas variables están por ejemplo los parámetros de los controles PID, lo que ayudara al operario en el ajuste y puesta en marcha.

(TODO Formatos de archivos)