



## **ARQUITECTURA DE SOFTWARE**

**"Control de un variador de frecuencias para una bomba de  
cavidad progresiva"**

PROFESOR	Martín Morixe
ALUMNOS	Federico Treuque, Ignacio Egea y Gastón Zales

## ÍNDICE

<b>ÍNDICE</b>	<b>2</b>
<b>Introducción</b>	<b>3</b>
<b>Objetivos e implementación</b>	<b>4</b>
<b>Requerimientos</b>	<b>4</b>
Requerimiento 1	4
Requerimiento 2	5
Requerimiento 3	5
Requerimiento 4	6
<b>Descripción del sistema mediante Vistas 4+1:</b>	<b>6</b>
Vista Lógica	6
Vista de desarrollo	8
Vista Física	13
Vista de procesos	14
Escenario/casos de uso	14
<b>Interfaz gráfica</b>	<b>15</b>
Ingreso de parámetros y elección de aceleración del sistema	15
Guardado y actualización de parámetros.	16
Puesta en marcha	17
Alcance de velocidad objetivo	18
Cambios de velocidad objetivo	19
Paro de emergencia	19
<b>Conclusiones</b>	<b>20</b>
<b>Anexo 1: Link al repositorio con código fuente</b>	<b>21</b>

## Introducción

Este sistema fue desarrollado con el fin de realizar el control de una bomba de cavidad progresiva, cuya función es la extracción de petróleo crudo de un pozo. Este programa otorga al usuario final la capacidad de hacer ajustes de velocidad, tiempos, escalas y monitorear cada etapa. Dicho sistema fue pensado específicamente para el control de bombas eléctricas trifásicas, para el cual se necesita de la comunicación con un variador de frecuencias, el cual se encarga de controlar la velocidad de bombeo.



El variador de frecuencias (VFD, del inglés: *Variable Frequency Drive* o bien AFD *Adjustable Frequency Drive*), toma control de la velocidad rotacional de un motor de corriente alterna (AC) por medio del control de la frecuencia de alimentación suministrada a dicho motor. Estos dispositivos son una herramienta esencial en muchas operaciones industriales, proporcionando una serie de ventajas que reducen el tiempo y los recursos necesarios para realizar las operaciones, aumentan la seguridad de los equipos y operarios, y ayudan a reducir los costes eléctricos, prolongando la vida útil de los motores.



La forma que el usuario controla un variador de frecuencia es accediendo a las direcciones de memoria del mismo, pudiendo así, acceder a registros de velocidad, tiempos, corriente, etc.. Una herramienta útil para realizar estos accesos son los monitores industriales, que consisten en paneles de control con una interfaz gráfica.



Programando con ayuda de algunas herramientas de software de ingeniería, se puede desarrollar un sistema con requerimientos específicos de usuario, como en este caso, el control de una bomba de cavidad progresiva. Este instrumento trabaja lodos de alta viscosidad, por lo cual se necesita de un control preciso del bombeo. Este bombeo, por lo general, es lento y se deben implementar variaciones no bruscas para evitar que trabaje en vacío.

## Objetivos e implementación

Los objetivos principales del proyecto consisten en:

- Implementar el programa para control del VDF sobre un framework en C++.
- Utilizar algún patrón de diseño de software que se ajuste a las necesidades.
- Documentar el diseño con lenguaje UML, utilizando las vistas 4+1 de Kruchten.
- Utilizar un sistema de control de versiones para llevar el código adelante.

Se utilizó el framework QT-CREATOR para la programación del sistema, ya que está orientado al desarrollo de interfaces gráficas e implementa C++.

Se hizo uso de los patrones de diseño de software Command y Singleton, explicando su implementación en el documento.

Se diseñaron varios diagramas en UML correspondientes a las vistas 4+1, como el diagrama de clases, estados, actividades, casos de uso, paquetes y despliegue.

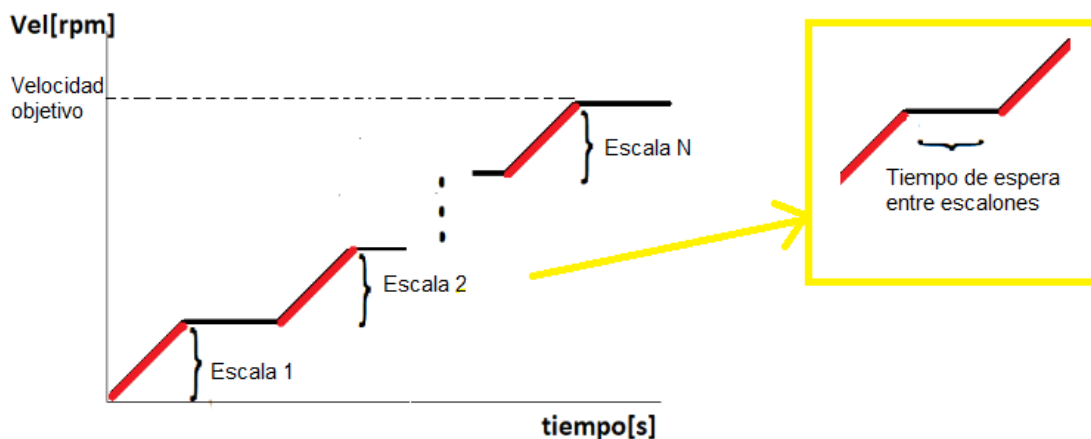
El código se controló con GIT, utilizando específicamente la plataforma Github.

## Requerimientos

Con respecto a los requerimientos que debe cumplir nuestro programa, estos consisten en:

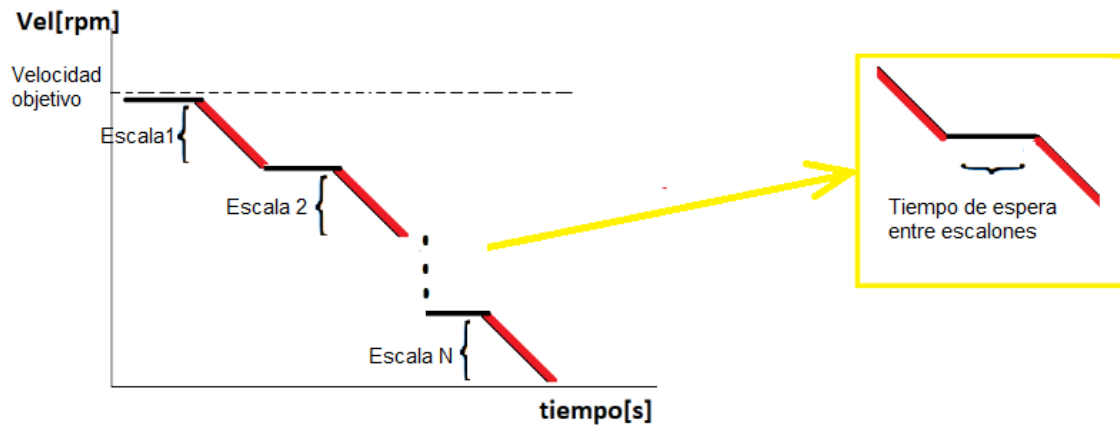
### Requerimiento 1

Se requiere un sistema que aumente la velocidad de bombeo de forma pausada y escalonada, para evitar que se trabaje en vacío. La cantidad de escalas deberán variar en función al material que se está bombeando. Deberá existir un tiempo de espera entre escalas, un parámetro importante para no forzar la bomba. Este se modificará en función de lo que se esté bombeando.



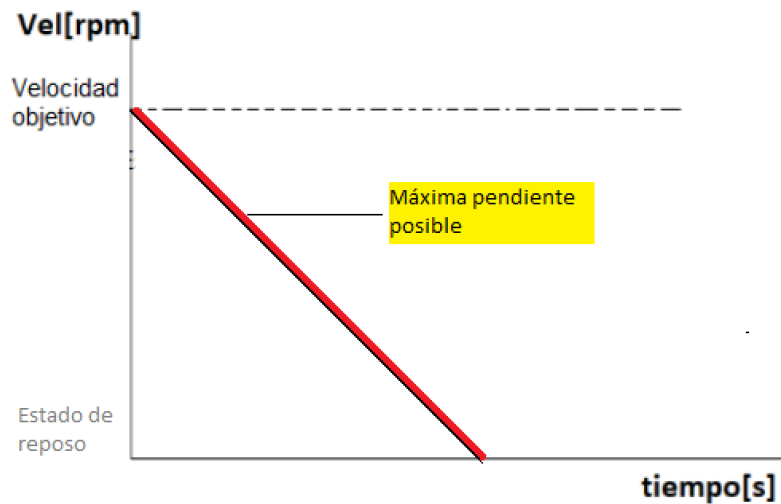
### Requerimiento 2

Se requiere una dinámica análoga para la desaceleración, con los mismo parámetros descritos para la aceleración.



### Requerimiento 3

Se requiere una parada de emergencia, para la cual el sistema, sin importar el estado donde se encuentre, guardará los datos actuales y comenzará a desacelerar hasta alcanzar el estado de reposo. Esta desaceleración a 0 RPM no será de forma escalonada, más bien en una sola escala y a la máxima pendiente posible..



#### Requerimiento 4

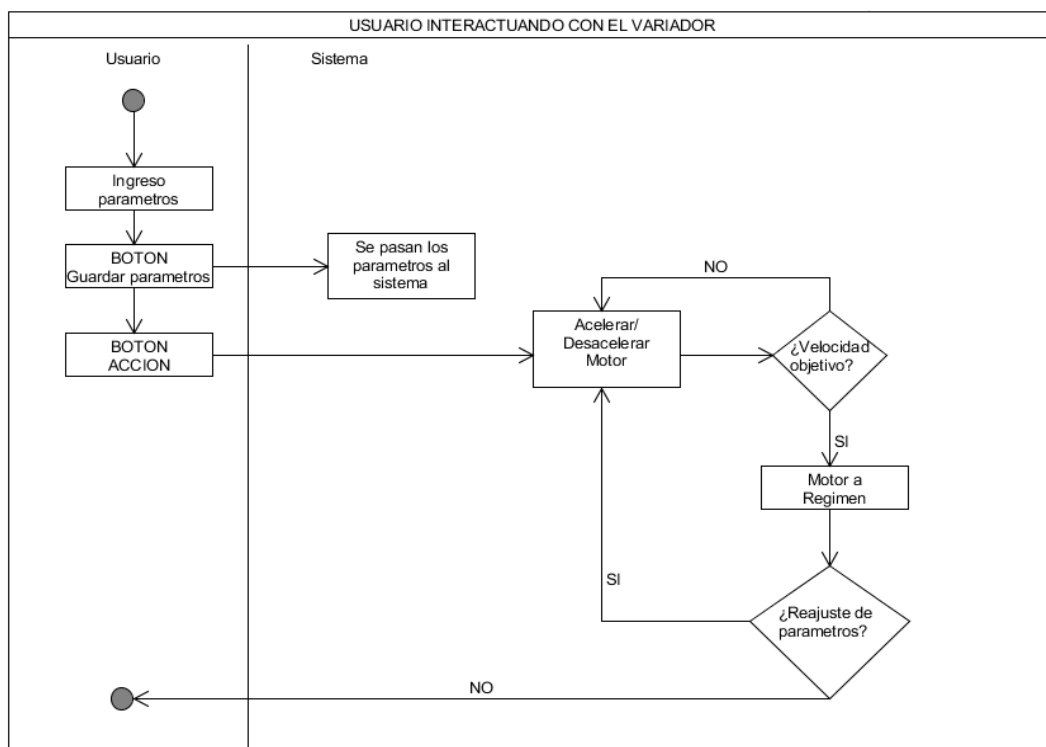
Se requiere desarrollar una interfaz gráfica donde el usuario pueda ajustar de forma manual la velocidad objetivo de la bomba, la cantidad de escalas, la velocidad con la que se desea pasar entre escalas, y el tiempo de espera entre ellas. Además, en dicha interfaz se deberá poder visualizar el estado actual del sistema, el estado actual de la bomba y sus RPM.

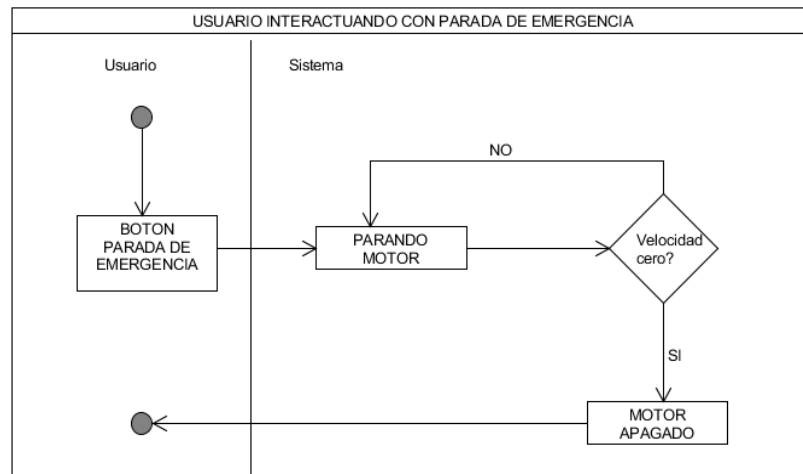
#### **Descripción del sistema mediante Vistas 4+1:**

##### Vista Lógica

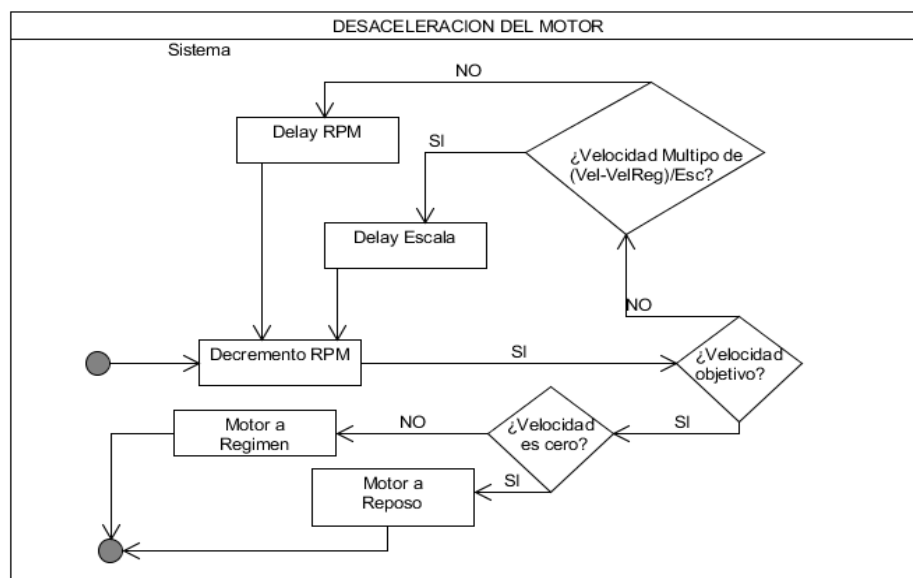
Mediante esta vista se pretende dar una descripción de la estructura y finalidad del diseño, con el objetivo de responder cuáles son las funcionalidades que el sistema provee al usuario final.

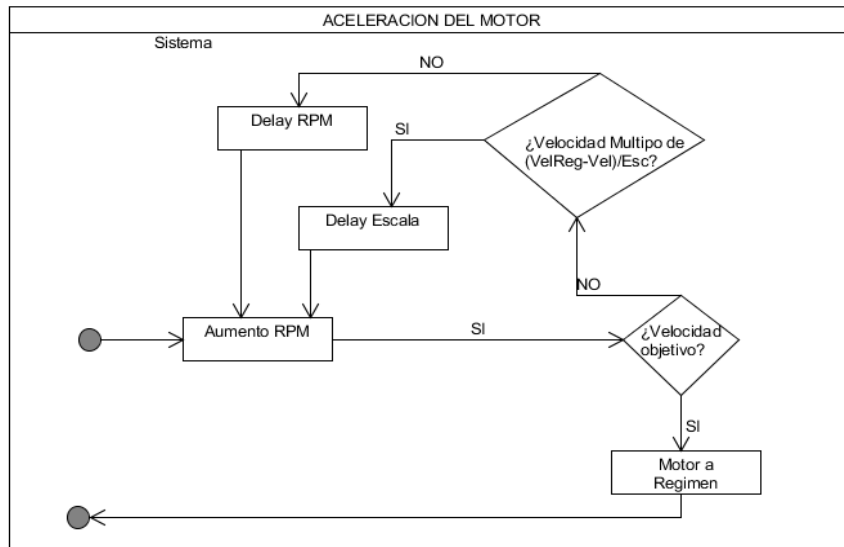
A continuación, mostramos los **diagramas de actividades** que describen la interacción del usuario con el sistema:





A continuación, mostramos los **diagramas de actividades** que describen cómo funciona la aceleración y desaceleración de la bomba :





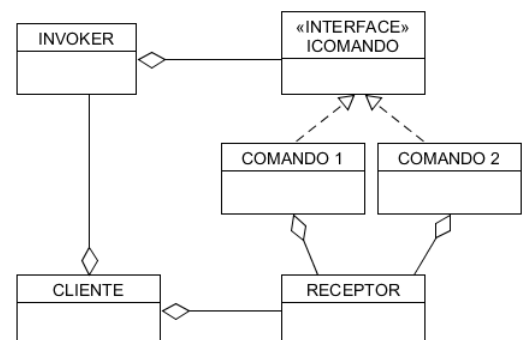
### Vista de desarrollo

En esta vista se pretenderá describir al sistema desde la perspectiva de un programador, y se ocupa de la gestión del software. En otras palabras, se mostrará cómo está dividido el sistema software en componentes y las dependencias que hay entre componentes.

Mediante el **diagrama de clases**, describiremos la estructura del sistema mostrando las clases con las que se compone, sus atributos, métodos y relaciones. En este sistema se implementó el patrón de diseño **COMMAND** y **SINGLETON**.

### Funcionamiento del Patrón COMMAND

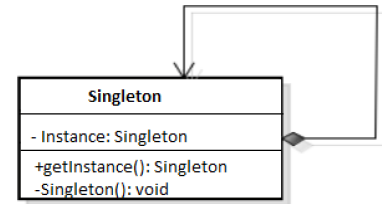
- **Comando:** Clase que ejecuta una función del receptor mediante su método Ejecutar(). Se implementan varias, una para cada función deseada.
- **Invoker:** Clase que se invoca con un comando como parámetro para ejecutarlo. Crea la distancia entre el cliente y el receptor, y le pide al comando que lleve a cabo la acción. Con su método Función() ejecuta el comando específico.
- **Cliente:** Clase capaz de llamar a los invocadores para ejecutar comandos.
- **IComando:** Interfaz cuyo único método es Ejecutar(), el cual es implementado de distinta manera por cada comando.
- **Receptor:** Clase que contiene las operaciones que necesitan ser llevadas a cabo por los comandos.





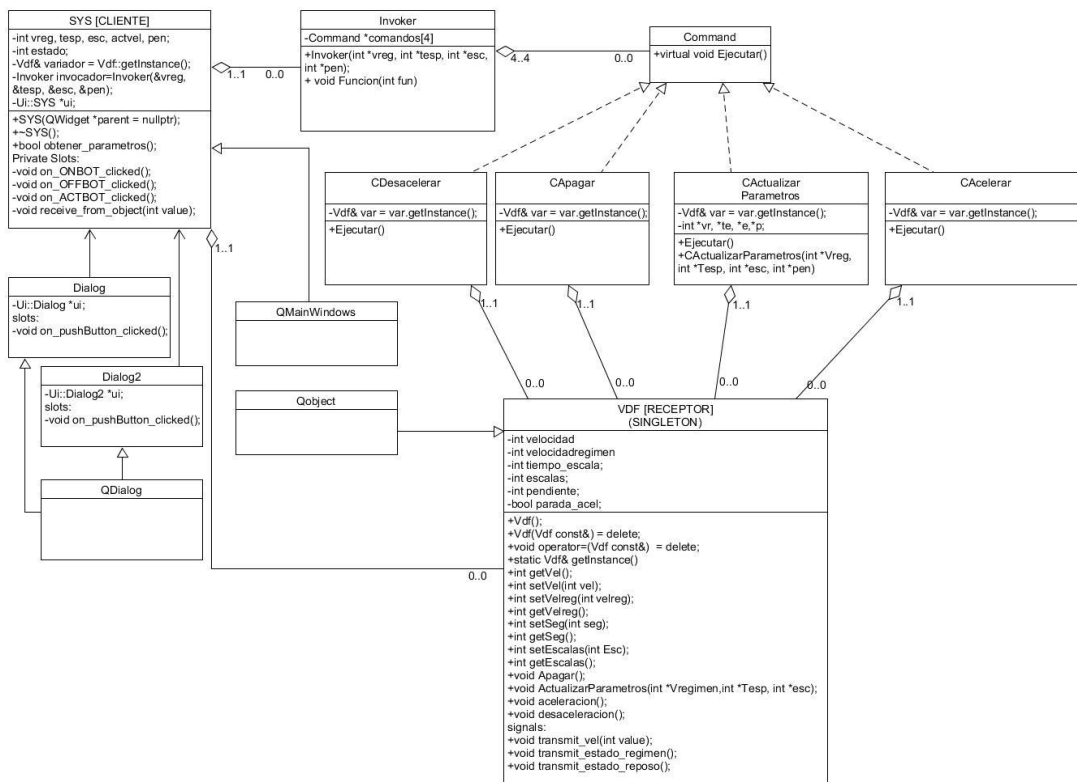
## Funcionamiento del Patrón SINGLETON

El patrón singleton consiste en permitir generar una instancia única de una clase y restringir la creación de más objetos. Se debe proporcionar un punto de acceso global a dicha instancia mediante un método público de la clase.



Este patrón permite generar una suerte de objeto global, al cual se puede acceder de forma global por las clases que lo conozcan. De esta forma, se garantiza que las clases que interactúan con esta clase singleton, hagan referencia siempre al mismo objeto.

## Diagrama de clases con patrones implementados



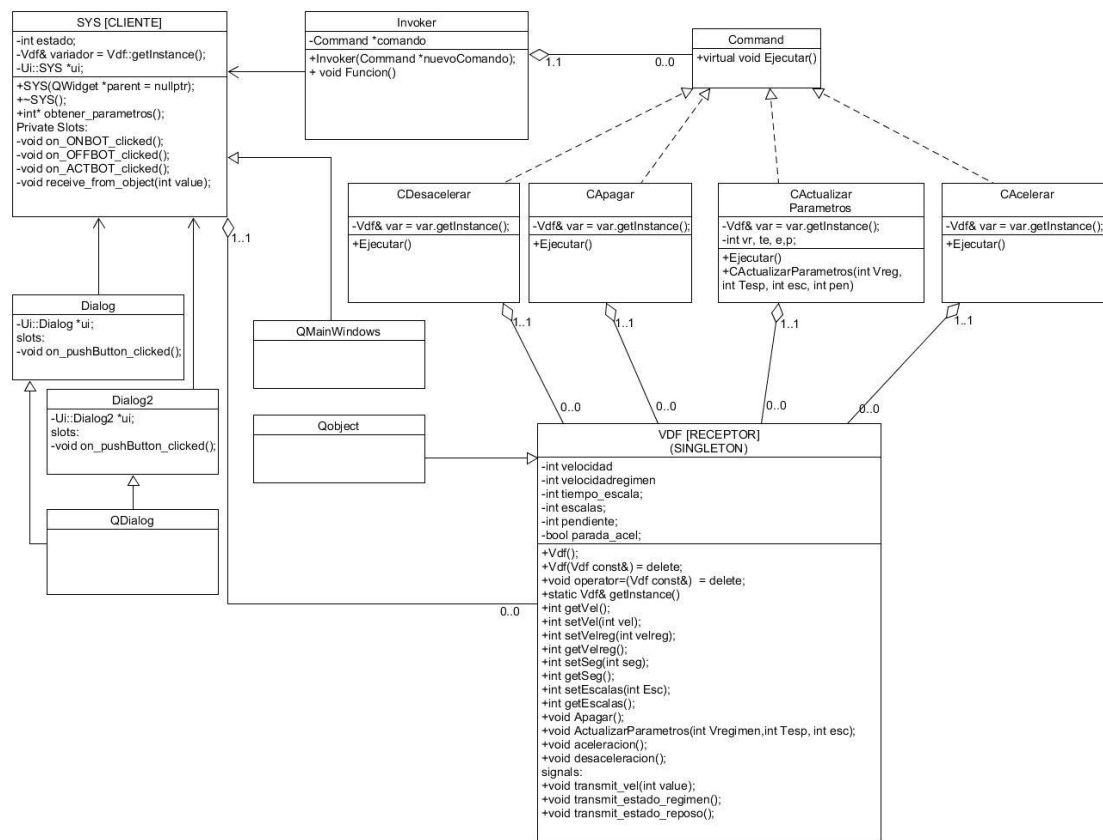
En el siguiente diagrama de clases se implementa el patrón Command de forma tal que existe un solo invocador dentro de la clase SYS, el cual tiene como atributos los 4 comandos necesarios para comandar el VDF dentro de un array. La forma en la que se ejecuta un comando es llamando al invocador y utilizando su método Función(), el cual recibe como parámetro una posición del array, dependiendo de qué comando se quiera ejecutar.

La clase SYS es la que tiene acceso al invocador, y por ende controla la ejecución de los comandos. Todos los comandos hacen referencia a una sola instancia de la clase VDF, haciendo de esta última un Singleton.

La clase SYS conoce a la instancia de VDF para acceder a los parámetros más esenciales del variador, como la velocidad actual y la de régimen. De esta forma, puede invocar al comando indicado dependiendo del estado del sistema, y definir el estado actual de la bomba.

El constructor del invocador recibe como parámetro las referencias a las variables globales que actualizan los valores del VDF. De esta forma, cuando se desea actualizar los parámetros, se modifican estas variables y el comando CActualizar se las asigna al variador.

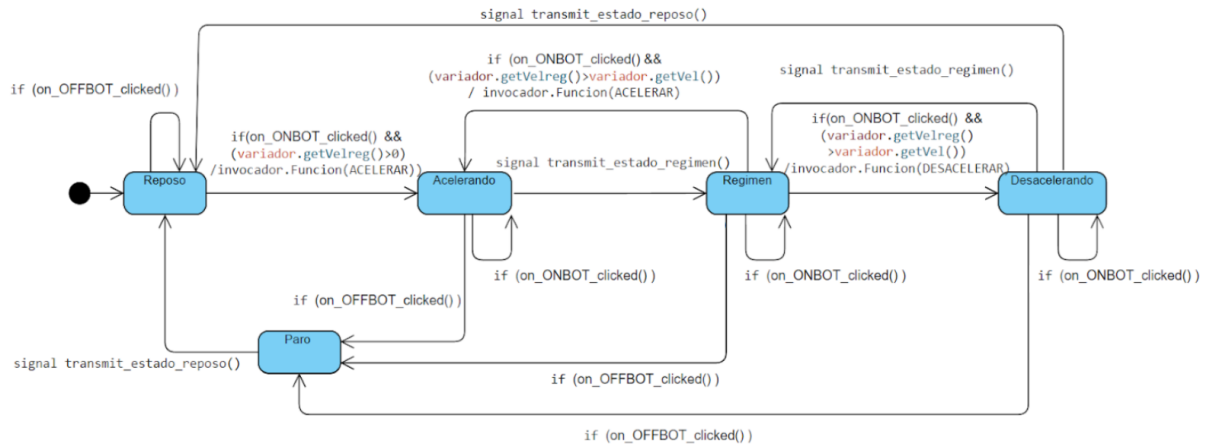
## Segunda implementación



En esta implementación existen múltiples instancias del invocador, el cual esta vez recibe como parámetro un solo comando para su construcción. Nuevamente, SYS tiene acceso a generar instancias del invocador, dependiendo del comando que se busque ejecutar. Para ejecutar un comando específico, se llama al atributo Función() del invocador específico en cuestión.

Una diferencia importante con la implementación anterior es, que los parámetros que se utilizaban para la inicialización del comando CActualizar no son más globales, sino que se le mandan al comando CActualizar como parámetro cuando es creado.

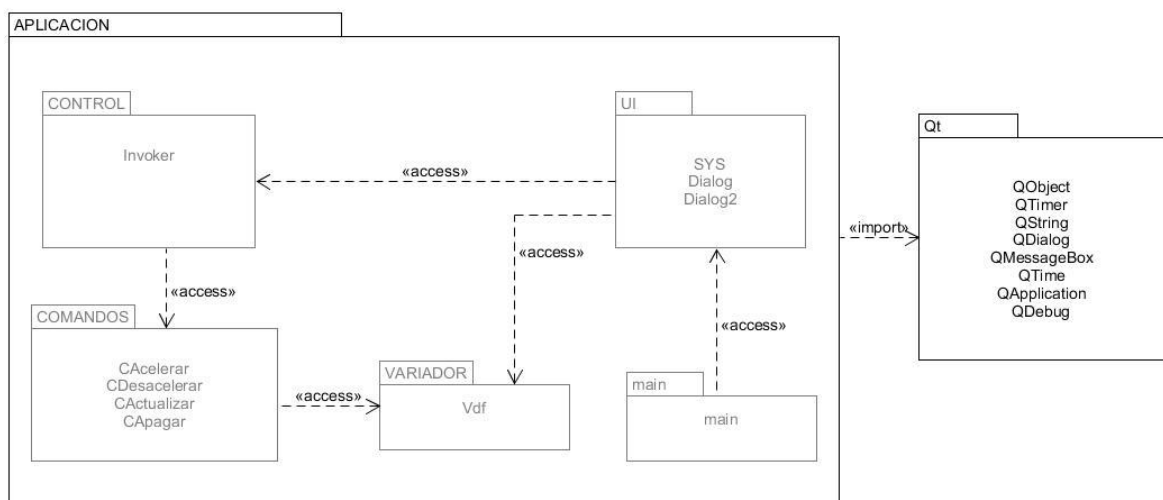
Utilizamos un **diagrama de estados** para describir el comportamiento completo del sistema frente a interacciones con la interfaz de usuario. Mediante el uso de una variable de estados definida dentro de la interfaz de usuario, logramos monitorear el comportamiento de la bomba en sus distintas etapas.



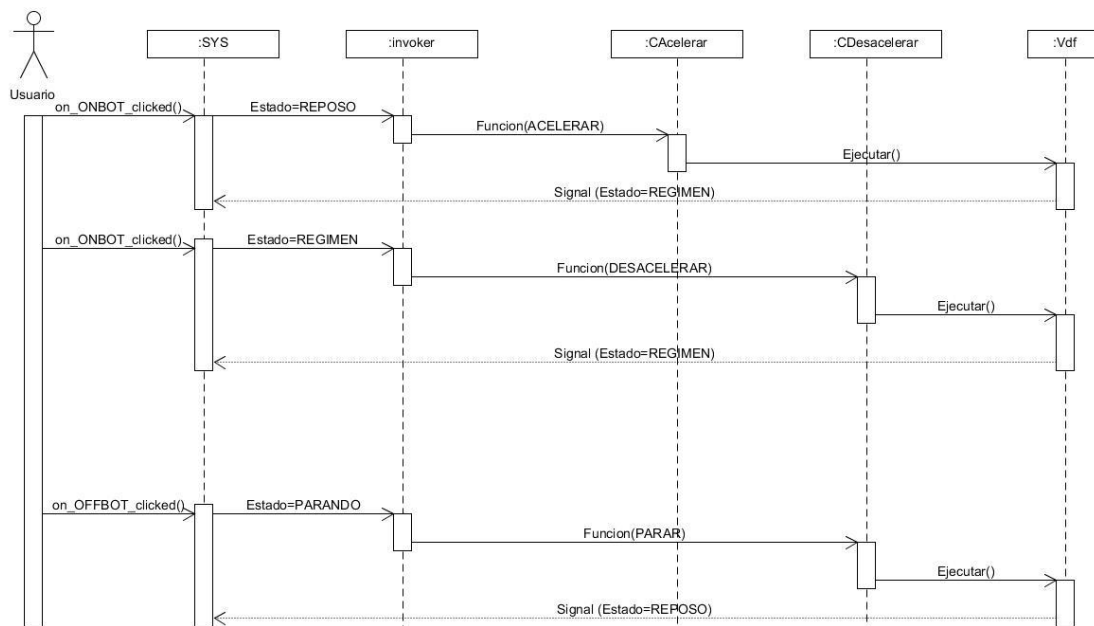
Descripción de los estados:

- **Estado REPOSO:** Estado donde el sistema permanece sin acciones a rpm 0.
- **Estado ACELERANDO:** Estado donde el variador incrementa las RPM del motor.
- **Estado DESACELERANDO:** Estado donde el variador decrementa las RPM del motor.
- **Estado REGIMEN:** Estado donde el variador deja de incrementar/decrementar las RPM del motor dejándolo a unas RPM fijas.
- **Estado PARO:** Estado donde el variador baja las RPM del motor hasta apagarlo.

Dada la interacción de la aplicación con las clases y con el uso de la librerías de QT, realizamos un **diagrama de paquetes** que describe la agrupación de clases según su función en el sistema, y cómo acceden entre ellas.



Para explicar el patrón COMMAND implementado en el sistema, graficamos un **diagrama de secuencia** pasando por la aceleración, desaceleración y parada del sistema. Se describe cada instancia de las clases involucradas, y cómo estas interactúan entre ellas cuando el usuario realiza una petición.

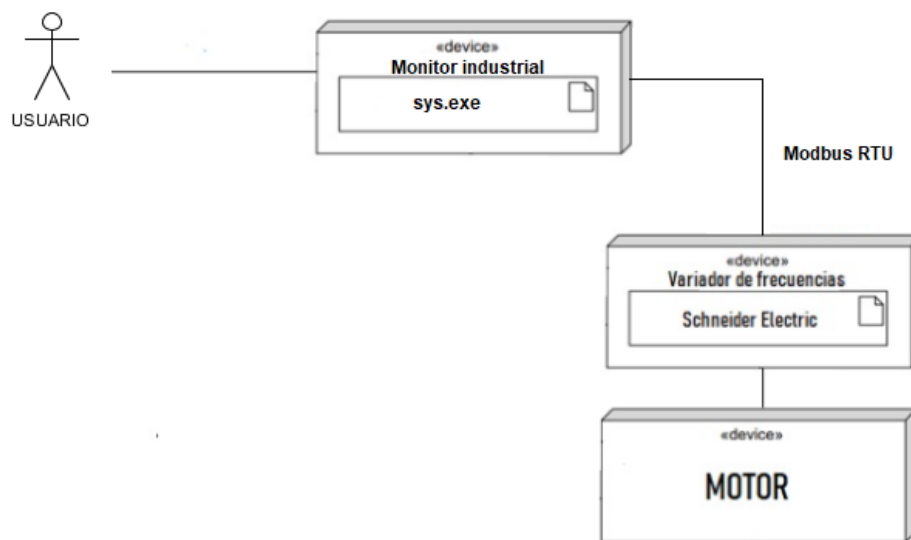


Las funciones Signal son métodos exclusivos del framework QT-CREATOR, que permiten la comunicación entre dos objetos del sistema. En este caso, se utilizan para que el VDF avise cuando finaliza la aceleración o desaceleración de la bomba. Al recibir la señal, SYS ejecuta una función que modifica la variable de estado del sistema.

### Vista Física

Esta vista describe el sistema desde la perspectiva de un ingeniero en sistemas, visualizando todos los componentes físicos que componen al proyecto, así como las conexiones físicas entre los dispositivos que conforman la solución.

Mediante la relación que tienen los diferentes componentes del sistema físico realizamos un **diagrama de despliegue**, que describe el hardware necesario para el diseño, como además el software que corre dentro de cada uno.



La interfaz Modbus RTU no fue implementada en el diseño, sino que se simuló tener un variador conectado dentro de la aplicación.

### Vista de procesos

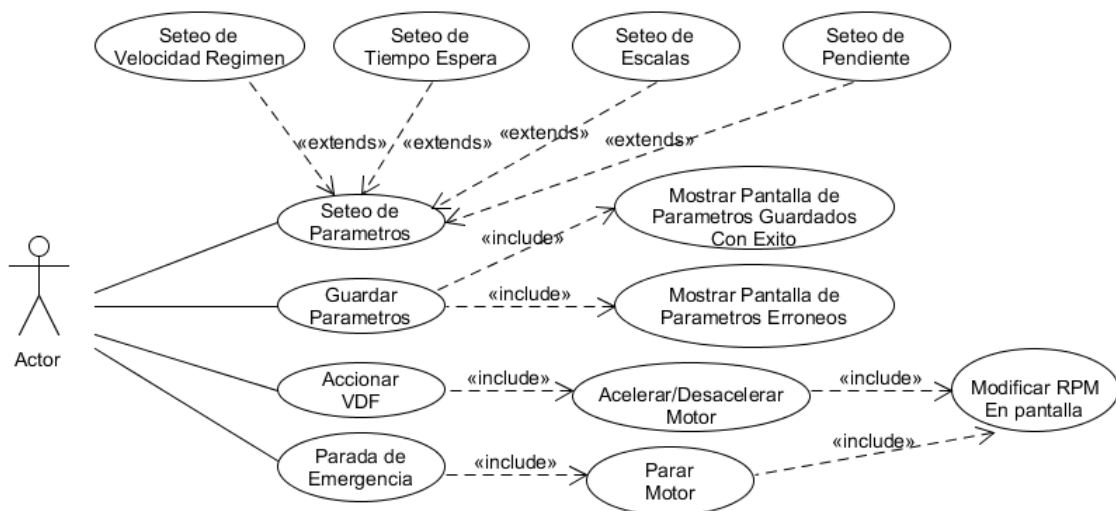
La vista de procesos explica los aspectos dinámicos del sistema, y se enfoca en el comportamiento que tiene el sistema en tiempo de ejecución. Explica cómo se comunican los procesos y describe su concurrencia, distribución, integración, performance y escalabilidad.

Al no haber implementado hilos de ejecución, no tenemos un diagrama para describir esto.

### Escenario/casos de uso

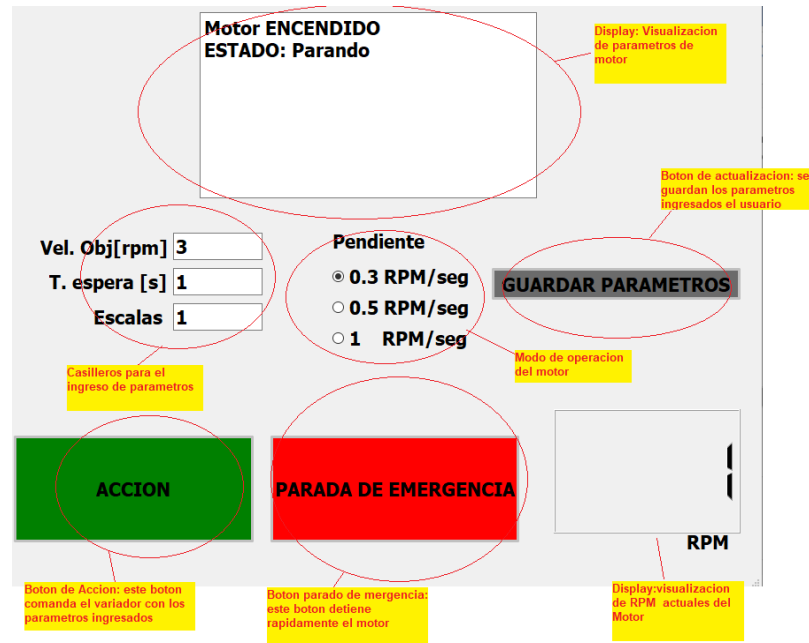
En esta vista se representarán los casos de uso del software, con el objetivo de relacionar las otras cuatro vistas ya descritas.

En este caso, utilizamos un **diagrama de casos de uso**, que implica una descripción gráfica de un posible usuario interactuando con el sistema.



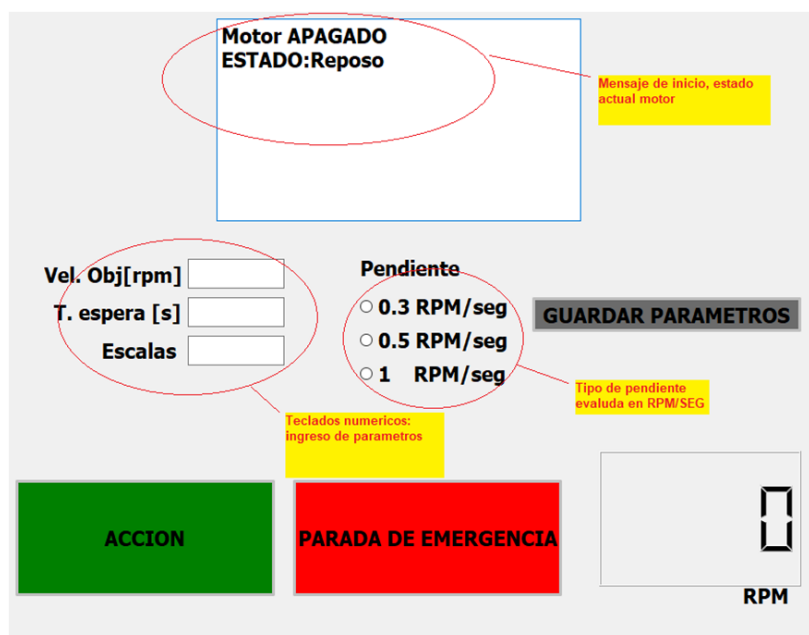
## Interfaz gráfica

Se diseñó una interfaz gráfica de manera que cumpla como principal objetivo proporcionar un entorno visual sencillo para permitir la comunicación directa entre el usuario y el sistema.



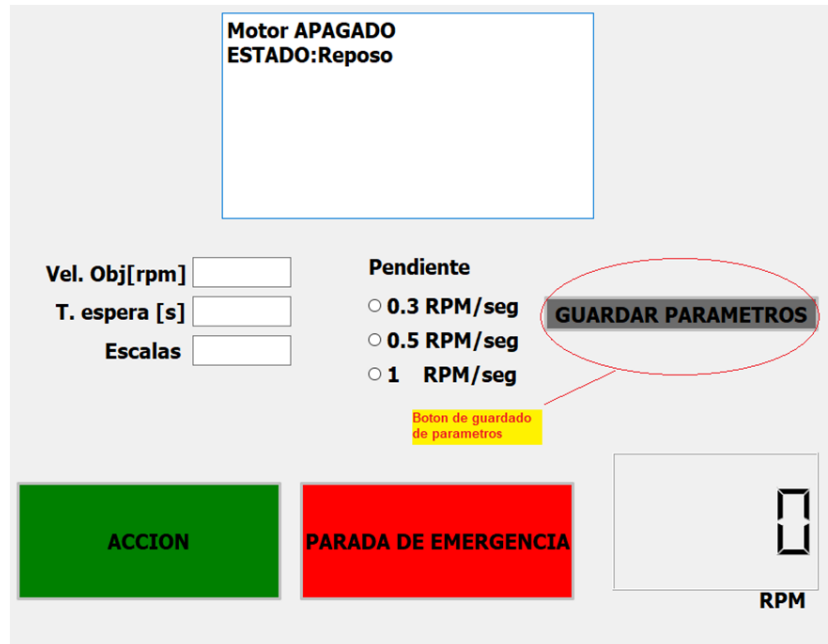
## Ingreso de parámetros y elección de aceleración del sistema

Al inicio del programa, el software mostrará un mensaje indicando el estado del motor, y estará a la espera del ingreso de parámetros, elección de la aceleración

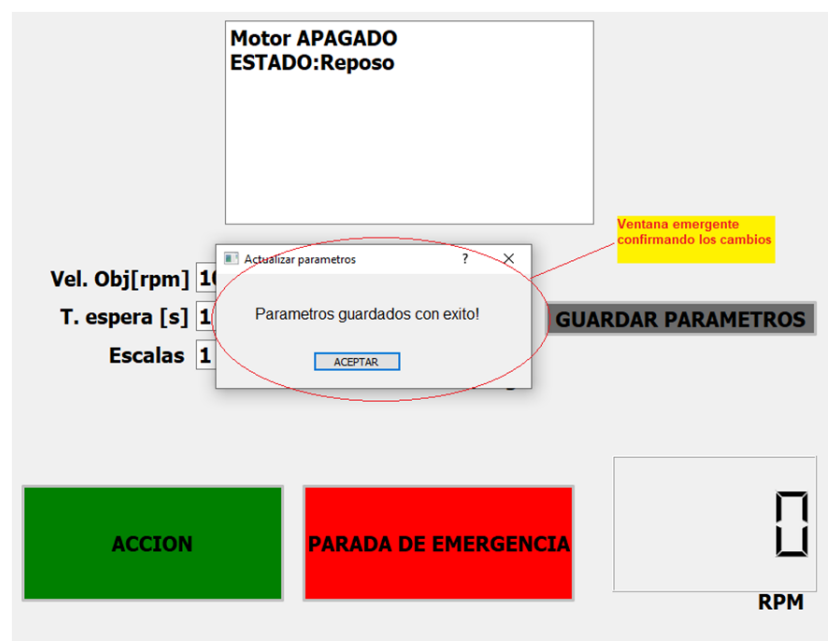


### Guardado y actualización de parámetros.

Una vez que el usuario ingrese los parámetros y eligió el tipo de aceleración del sistema, el software quedará esperando a que se guarden los parámetros.

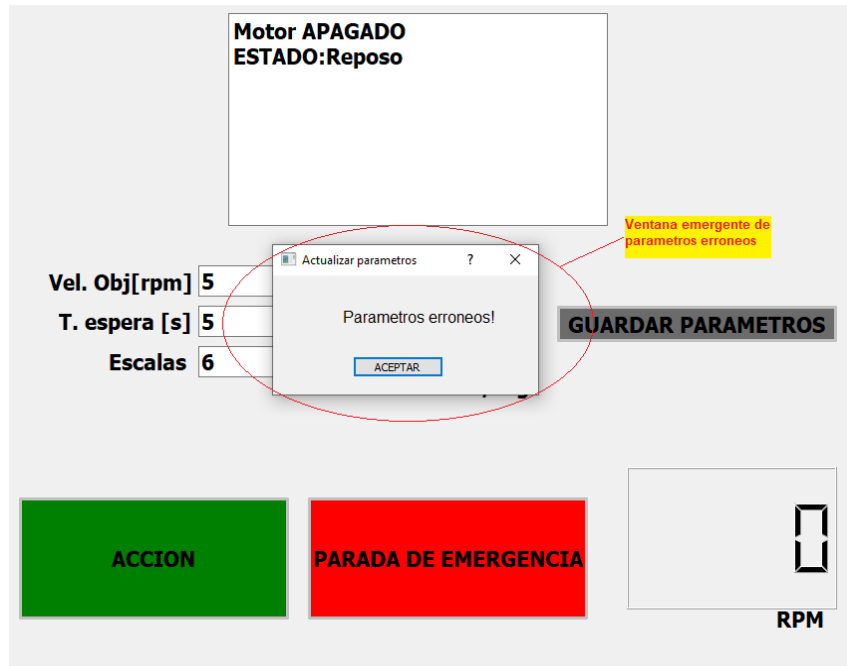


Una vez presionado el botón de guardado el sistema reconocerá los parámetros y la aceleración pretendida e imprimirá en pantalla confirmando que los parámetros han sido actualizados, además de una ventana emergente confirmando la actualización.





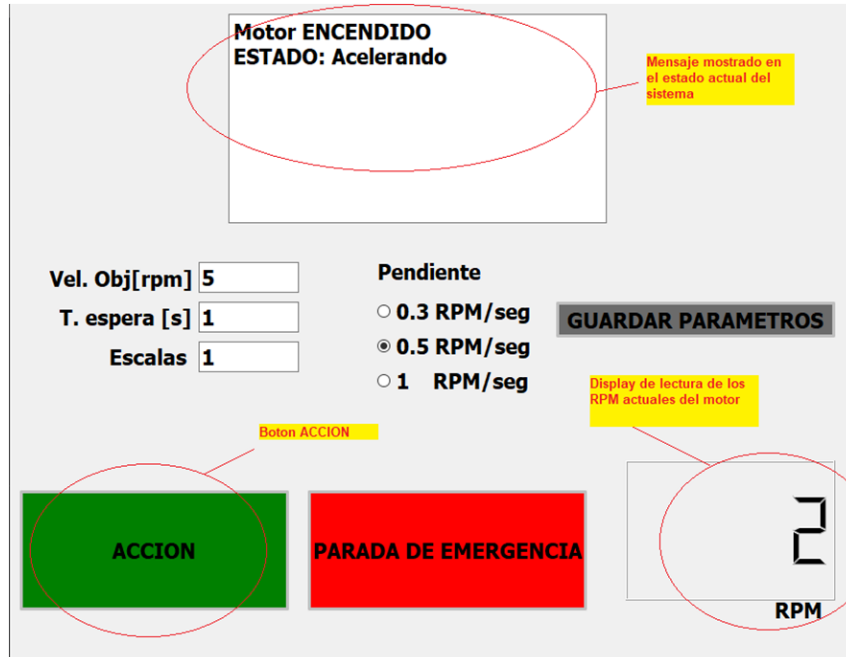
Si el usuario ingresa mal los parámetros el emergerá una ventana mencionando que los parámetros son erróneos y no guardará los cambios hasta que el usuario ingrese los parámetros correctamente.



### Puesta en marcha

Una vez actualizados los datos que necesita el sistema para su funcionamiento, solo quedará por darle marcha y lograr la velocidad objetivo requerida. Cuando se presione el botón de marcha, el sistema imprimirá un mensaje confirmando la solicitud, y habrá un segundo display en el cual se visualizarán las RPM leídas por el VDF.

**NOTA:** Para que el sistema se ponga en marcha si o si es necesario haber guardados los parámetros ingresados y el tipo de aceleración elegida por el usuario, en caso contrario el sistema ignorará el pedido de marcha.



**Motor ENCENDIDO**  
**ESTADO: Acelerando**

Mensaje mostrado en el estado actual del sistema

Vel. Obj[rpm]   
 T. espera [s]   
 Escalas

Pendiente  
☐ 0.3 RPM/seg  
☒ 0.5 RPM/seg  
☐ 1 RPM/seg

GUARDAR PARAMETROS

Display de lectura de los RPM actuales del motor

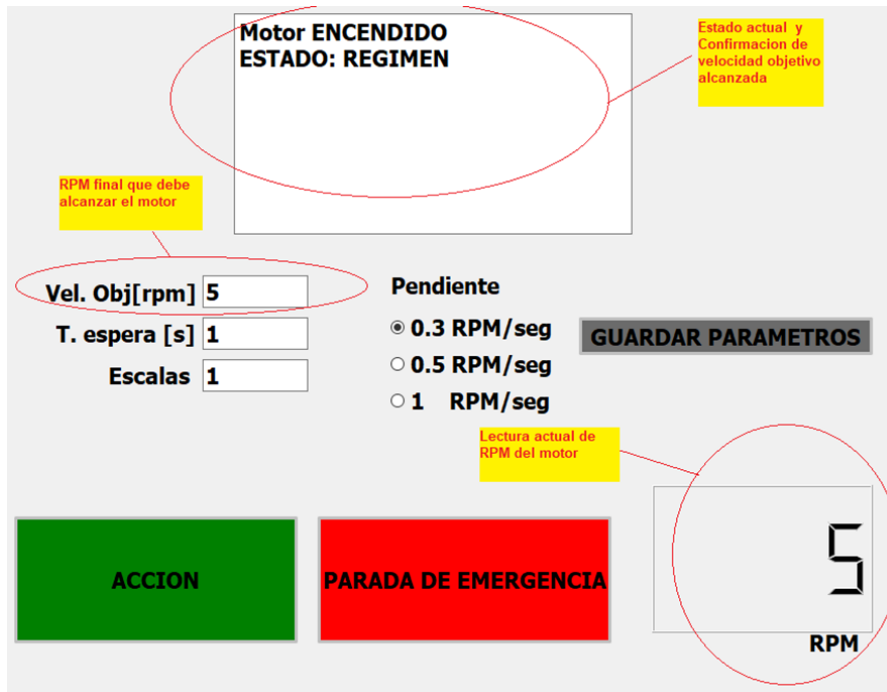
Boton ACCION

**ACCION** **PARADA DE EMERGENCIA**

2  
RPM

### Alcance de velocidad objetivo

Una vez alcanzada la velocidad objetivo el sistema imprimirá un mensaje confirmando el estado de “Régimen”.



**Motor ENCENDIDO**  
**ESTADO: REGIMEN**

Estado actual y Confirmacion de velocidad objetivo alcanzada

RPM final que debe alcanzar el motor

Vel. Obj[rpm]   
 T. espera [s]   
 Escalas

Pendiente  
☒ 0.3 RPM/seg  
☐ 0.5 RPM/seg  
☐ 1 RPM/seg

GUARDAR PARAMETROS

Lectura actual de RPM del motor

**ACCION** **PARADA DE EMERGENCIA**

5  
RPM

### Cambios de velocidad objetivo

Una vez que el sistema alcanza la velocidad objetivo, el usuario tiene la posibilidad de poder cambiar y actualizar los parámetros del motor. De esta manera el sistema acelera o desacelera en caso de ingresar un mayor o menor velocidad objetivo respectivamente. En ambos casos se deberán repetir los pasos:

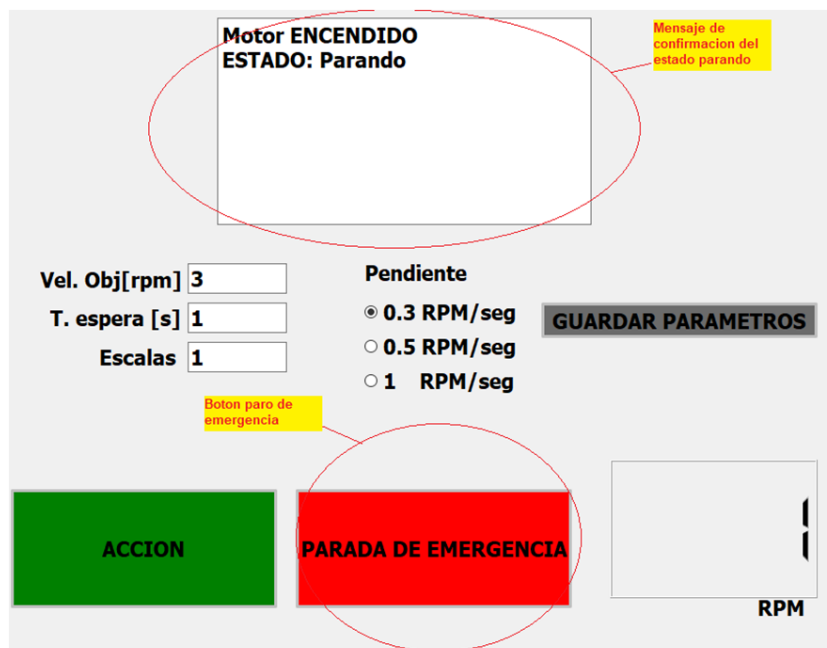
- **Ingreso de parámetros y elección de aceleración del sistema.**
- **Guardado y actualización de parámetros.**

**NOTA:** En caso de no ingresar nuevos parámetros y un nuevo tipo de aceleración el sistema reconocerá como parámetros los datos anteriormente guardados.

Una vez ingresada la nueva velocidad objetivo el sistema imprimirá un mensaje tanto en el caso de ACELERAR como en el caso de DESACELERAR hasta lograr la velocidad objetivo requerida.

### Paro de emergencia

La utilidad de esta función es que no importa el estado en el que se encuentre el sistema, al presionar el botón de paro se imprimirá un mensaje confirmando la petición y el sistema inmediatamente comenzará a desacelerar hasta lograr el estado de reposo del sistema.



The screenshot displays a control interface for a motor system. At the top, a status box indicates "Motor ENCENDIDO" and "ESTADO: Parando". Below this, there are input fields for "Vel. Obj[rpm]" (set to 3), "T. espera [s]" (set to 1), and "Escalas" (set to 1). To the right, under the heading "Pendiente", there are three radio button options for acceleration rates: "0.3 RPM/seg" (selected), "0.5 RPM/seg", and "1 RPM/seg". A "GUARDAR PARAMETROS" button is located next to these options. At the bottom left, there is a green "ACCION" button and a red "PARADA DE EMERGENCIA" button. A yellow callout points to the emergency stop button, labeling it "Boton paro de emergencia". Another yellow callout points to the status box, labeling it "Mensaje de confirmacion del estado parando". On the bottom right, there is a small display area labeled "RPM" showing a vertical scale.

## **Conclusiones**

Llevar a cabo el sistema nos pareció un gran desafío debido a que requirió plasmar los conocimientos adquiridos la cursada de la materia, trabajar en equipo con un control de versiones y complementar la capacidad de análisis de cada uno de los integrantes, llevando así a elegir la solución más óptima para el proyecto .

Mediante herramientas como QT-CREATOR, lenguaje de programación C++, la utilización de patrones de diseños y diagramas de UML, se logró con satisfacción la cumplir los objetivos propuestos.

Fue importante considerar el desarrollo de un sistema lo más robusto posible, garantizando la estabilidad y su correcto funcionamiento, evitando así cualquier tipo de inconvenientes al usuario final.

## Anexo 1: Link al repositorio con código fuente

<https://github.com/Proyecto-Arquitectura-de-Software-UNRN/Control-variador-de-frecuencia>

Nota: La rama **main** contiene el código correspondiente a la primera implementación del patrón COMMAND, correspondiente al software descrito en la totalidad del informe. La rama **2da-implementación-patrón-Command** contiene el código correspondiente a la segunda implementación.