Facultad 4

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Componentes gráficos para la representación de motores eléctricos en el SCADA SAINUX 2.0

Autor: Alejandro Rufino Cabo Marchena

Tutor: Ing. Adrián Carmona Hernández

Co-Tutora: Ing. Tahumara González Galbán

Declaración de autoría

Por este medio declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas (UCI) para que hagan el uso que estime pertinente con este trabajo.

Para que así conste firmo la presente a los ____ días del mes _____ del 2018.

_____

Firma del autor: Alejandro Rufino Cabo Marchena

_____

Firma del tutor (a): Ing. Adrián Carmona Hernández

_____

Firma del tutor (a): Ing. Tahumara González Galbán

Síntesis de los tutores

Tutor: Ing. Adrián Carmona Hernández

Síntesis: Graduado de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas (UCI). Cuatro años de experiencia en el desarrollo de sistemas *SCADA*. Desarrollador de los módulos Adquisición de Datos e Interfaz Hombre Máquina en el sistema SAINUX 2.0.

Correo electrónico: acarmona@uci.cu

Co-Tutor: Ing. Tahumara González Galbán

Síntesis: Graduada de Ingeniera en Ciencias Informáticas en la Universidad de las Ciencias Informáticas (UCI). Tres años de experiencia como Especialista principal de Gestión de Proyectos en el CEDIN.

Correo electrónico: tahumara@uci.cu

Dedicatoria

A mis padres:

Por haberme apoyado en todo momento, por sus consejos, sus valores, por la motivación constante que me ha permitido ser una persona de bien, pero más que nada, por su amor.

Agradecimientos

Primeramente, darle gracias a mis tutores, los cuales son los mejores que pude haber aspirado en este trabajo de diploma, Tahumara y Adrián, sin ustedes no hubiera sido posible este trabajo. A Adolfo, que, aunque oficialmente no es mi tutor me ha enseñado y guiado durante todo el desarrollo de este trabajo y me ha ayudado con cada uno de los percances.

Le agradezco a mi madre, por haberme acompañado y guiado a lo largo de mi carrera, por ser mi fortaleza en los momentos de debilidad y por brindarme una vida llena de aprendizajes, experiencias y sobre todo felicidad. Por apoyarme en todo momento, por los valores que me han inculcado, y por haberme dado la oportunidad de tener una excelente educación en el transcurso de mi vida.

A mi padre, aunque físicamente no esté conmigo, ha sido un excelente ejemplo de vida a seguir, quien me inculcó las enseñanzas que me impulsaron a estar hoy aquí frente este auditorio y superarme tanto físico como intelectualmente día a día.

A mi queridísima novia Danay, que sin ella hubiera sido muy difícil estar aquí ahora, gracias por el apoyo y las fuerzas que me has brindado incondicionalmente y soportar quien soy y como soy. Gracias por los consejos, por el cariño cuando más lo he necesitado y por todo el tiempo que me has dedicado escuchando mis problemas y ayudándome a resolverlos.

A mis amigos, a todos, que de una forma u otra me ayudaron en el transcurso de la carrera a superarme y ser mejor persona. A Oscar Luis, René, Deborah, Abel, Carlos, Ernesto, David, Gustavo y todos los que han estado allí en las buenas y en las malas.

A mis mentores, Joel y Sergio, por sus consejos y enseñanzas, sin ustedes no podría ser lo que soy ahora, gracias por su apoyo en los momentos que los he necesitado.

Resumen

Los Sistemas de Supervisión, Control y Adquisición de Datos (*SCADA*, por sus siglas en inglés) son utilizados para supervisar, controlar y adquirir datos de procesos industriales. El Centro de Informática Industrial (CEDIN), perteneciente a la facultad 4 de la Universidad de las Ciencias Informáticas (UCI) cuenta con el *SCADA* SAINUX 2.0, el cual a partir de su Interfaz Hombre-Máquina (*HMI*, por sus siglas en inglés) se encarga de brindar una interacción con el sistema al operario sobre los procesos para garantizar su control total. Actualmente SAINUX 2.0 está orientado mayormente al sector petrolero mediante la representación de componentes gráficos tales como tuberías, válvulas, bombas y tanques para el almacenamiento del crudo. Sin embargo, un sector en el que se podría introducir el sistema es el eléctrico, pero no se cuenta con los componentes para representar una red o circuito eléctrico. Este trabajo tuvo como objetivo el desarrollo de componentes gráficos de tipo motores eléctricos para el *HMI* del *SCADA* SAINUX 2.0, lográndose como resultado una paleta de diez componentes sujetos a las especificaciones y estándares internacionales emitidos para este tipo de sinóptico gráfico. Lo que permitió la correcta representación de los motores eléctricos, posibilitando su reutilización para varios despliegues a partir de las potencialidades de la extensión de los vectores gráficos escalables (*SVG,* por sus siglas en inglés), utilizada en la solución.

Palabras claves: componente gráfico, *HMI*, motores eléctricos, *SCADA,* vector gráfico escalable.

Índice

Índice de figuras

Índice de tablas

Introducción

En la Universidad de las Ciencias Informáticas (UCI), se ha trabajado intensamente en el apoyo, la gestión y el desarrollo en diferentes sectores del ámbito social como la salud, la educación y el deporte. Contribuyendo a la informatización de la sociedad cubana, a partir de la utilización de las Tecnologías de la Información y las Comunicaciones (TIC), para lograr así una mayor generación de conocimiento y riquezas.

En el amplio campo de la informática uno de los eslabones fundamentales es la automatización de procesos industriales la cual surge a partir de la necesidad de recolectar, almacenar y visualizar la información de estos procesos. Los Sistemas de Supervisión, Control y Adquisición de Datos (*SCADA*, por sus siglas en inglés) son un ejemplo de automatización, diseñados con la finalidad de controlar y supervisar procesos a distancia, basándose en la adquisición de datos generados por dispositivos de campo. Estos sistemas pueden estar compuestos por módulos como manejadores, núcleo de procesamiento de datos, base de datos históricos y la Interfaz Hombre Máquina (*HMI*, por sus siglas en inglés) [1].

En el Centro de Informática Industrial (CEDIN) de la UCI se desarrolla el Sistema de Automatización Industrial UX 2.0 (SAINUX 2.0). Este sistema cuenta con un conjunto de componentes gráficos que permiten representar en la pantalla de un ordenador procesos industriales a distancia mediante un *HMI*. En su mayoría, los componentes gráficos con que cuenta el *SCADA* SAINUX 2.0 son orientados al sector petrolero, destacándose las tuberías, válvulas, bombas y tanques para el almacenamiento de crudo.

Los componentes gráficos son una representación digital de componentes reales, los cuales son utilizados en sistemas *SCADA* para la industria del petróleo, gas, minero y eléctrico; donde el operador puede interactuar con ellos asignándoles funciones que harían en un escenario real, mediante un editor gráfico en un *HMI*, configurando sus variables y comportamiento. En el caso de los componentes gráficos eléctricos, se rigen por dos estándares internacionales, la Comisión Electrotécnica Internacional (*IEC*, por sus siglas en ingles) y el Comité Europeo de Normalización Electrotécnica (CENELEC) [2].

Para aumentar la versatilidad del SAINUX 2.0 y así mismo pueda ser explotado en otros sectores donde existe gran demanda de sistemas *SCADA*, como por ejemplo el sector eléctrico, la implementación de los componentes gráficos de tipo motor eléctrico en dicho sector permitiría:

- ✓ Monitorear el estado de las líneas eléctricas y sus parámetros eléctricos.
- ✓ Representación de redes eléctricas, circuitos y símbolos de instrumentación.
- ✓ Aumento del valor agregado del sistema, siendo posible utilizarlo en más de un sector industrial.

La representación de redes eléctricas, circuitos de medición de intensidad de la electricidad y sistemas de cargas pesadas, se realiza mediante la amplia gama de símbolos de instrumentación eléctrica existentes, dentro de los mismos se encuentran los motores eléctricos, los cuales se utilizan para representar los equipos encargados de transformar energía eléctrica en energía mecánica por medio de interacciones electromagnéticas de sus distintos componentes internos. Aunque SAINUX 2.0 no cuente con los componentes gráficos, como los antes mencionados, la representación de los mismos es posible mediante el componente Imagen y agrupaciones de los componentes básicos como elipses, líneas, vectores y cuadrados, lo cual no sería una vía factible ya que limitaría la exportación y la escalabilidad de dichos componentes, por lo tanto la versatilidad del *SCADA* SAINUX 2.0 se encuentra limitada su implantación en sectores estratégicos para el país donde se necesiten monitorear parámetros eléctricos.

A partir de la situación problemática antes expuesta surge como **problema de investigación**: ¿Cómo contribuir a la representación de componentes gráficos de motores eléctricos en el *HMI* del *SCADA* SAINUX 2.0? Planteándose como **objeto de estudio** el desarrollo de componentes gráficos para la industria eléctrica mediante sistemas *SCADA*.

Para darle solución al problema planteado, se propone como **objetivo general:** Desarrollar componentes gráficos que permitan representar motores eléctricos en la Interfaz Hombre Máquina del *SCADA* SAINUX 2.0. Enmarcándose, así como **campo de acción,** la representación de componentes gráficos para la industria eléctrica en la Interfaz Hombre Máquina del Sistema *SCADA* para SAINUX 2.0.

Buscando darle cumplimiento al objetivo planteado surgen como **tareas de investigación:**

- ✓ Elaboración del marco teórico conceptual relacionado con los aspectos teóricos que sustentan la investigación.
- ✓ Realización del análisis y diseño de la propuesta de solución.
- ✓ Desarrollo de la propuesta de solución como soporte al modelo.
- ✓ Realización de pruebas funcionales a la implementación.

Una vez obtenida la información referida a las tareas de investigación se plantean los **posibles resultados:** Componentes gráficos que permitan representar motores eléctricos en la Interfaz Hombre Máquina del *SCADA* SAINUX 2.0.

Diseño metodológico de la investigación:

Con el objetivo de agilizar el proceso de investigación y estudio de la información referente a la solución propuesta, se utilizó el método de investigación exploratoria con el objetivo de conocer e investigar Sistemas *SCADA* que utilicen componentes gráficos para el sector eléctrico.

Métodos utilizados en la investigación:

Métodos teóricos:

- ✓ **Histórico lógico:** permitió efectuar un estudio de los componentes eléctricos para su uso en la industria eléctrica y su evolución histórica en el mundo, así como profundizar los conocimientos sobre los sistemas de representación de motores eléctricos en dichos sistemas en los últimos años.
- ✓ **Modelación:** facilitó la creación de diagramas o modelos, que permiten la confección teórica de los motores eléctricos, haciendo más sencilla su comprensión.
- ✓ **Análisis-Síntesis:** posibilitó el estudio de los sistemas *SCADA* en partes separadas y las relaciones entre sus módulos, para así adquirir el conocimiento en concreto de dichos sistemas. También posibilitó el entendimiento analítico de dichos sistemas y cada uno de sus módulos para arribar a conclusiones precisas en función de la problemática planteada, sintetizado los criterios principales de cada uno de dichos módulos obtenidos de investigaciones previas.

Métodos empíricos:

- ✓ **Revisión y consulta de la información:** facilitó el análisis y procedimiento de las citas de toda la bibliografía consultada para la elaboración del marco teórico de la investigación.

El presente trabajo de investigación está conformado por tres capítulos con los siguientes temas a tratar:

Capítulo 1. “Fundamentación teórica”: En este capítulo se aborda el estudio y análisis de los lenguajes, metodologías, y herramientas posibles a usar para dar una solución a la problemática planteada, incluye además una descripción del estado del arte además de un estudio de sistemas similares, así como la evolución de los *SCADA*.

Capítulo 2. “Descripción de la solución”: En este capítulo se describe la propuesta de solución y se especifican los procesos esenciales que influyen en el objeto de estudio, se definen los requerimientos funcionales y no funcionales, los diagramas para el apoyo de la comprensión de su funcionamiento, así

como las fases del desarrollo de la solución, tomando como base la metodología *AUP*-UCI para la guía del proceso de desarrollo.

Capítulo 3. “Implementación y Prueba”: En este capítulo se aborda sobre la descripción de la implementación del sistema, se realizan los diagramas de despliegue y de componentes, se define el estándar de codificación y se realizan las pruebas pertinentes a las que fue sometida la aplicación.

Capítulo I. Fundamentación teórica

1.1 Introducción

El presente capítulo aborda los conceptos relevantes de la investigación, las tecnologías utilizadas, el lenguaje de programación empleado y herramientas explotadas para el diseño e implementación de los componentes a insertar en el SAINUX 2.0. Se describen características generales de los sistemas *SCADA*, los módulos involucrados en el mismo, así como los símbolos utilizados mundialmente para la representación de motores eléctricos para la industria.

1.1 Automatización de procesos industriales

La automatización de los procesos industriales, se aplica a sistemas de control y de tecnología informática para reducir la necesidad de la intervención humana en un proceso. En el enfoque de la industria, la automatización es el paso más allá de la mecanización en donde los procesos industriales son asistidos por máquinas o sistemas mecánicos que reemplazan las funciones que antes eran realizada por animales. Mientras en la mecanización los operadores son asistidos con maquinaria a través de su propia fuerza y de su intervención directa, en la automatización se reduce de gran manera la necesidad mental y sensorial del operador. De esta forma presenta grandes ventajas en cuanto a producción más eficiente y disminución de riesgos al operador [3].

Existen varias herramientas para la automatización de procesos industriales, tales como los sistemas *SCADA* y los Sistemas de Diseño Asistidos por Computadora (*CADS*, por sus siglas en inglés), ambos muy útiles para el correcto control y diseño de la infraestructura de cualquier sector de la industria. En los sectores como minería, petróleo, electricidad y gas, los *SCADA* juegan un desempeño crucial para la supervisión y el control del comportamiento de cada área donde se requiera un constante chequeo de lo que sucede y se encuentra distante de los especialistas que controlan estos sucesos. Por otra parte, los CADS son sumamente necesarios para el diseño de infraestructuras, equipos y herramientas para el trabajo, dando así menos margen al error humano en la creación de los componentes necesarios.

1.1.1 Ventajas de la aplicación de automatización de procesos industriales

- Reemplazo de operadores humanos en tareas repetitivas o de alto riesgo y en tareas que están fuera del alcance de sus capacidades como levantar cargas pesadas, trabajos en ambientes extremos o tareas que necesiten manejo de una alta precisión [3] .

- Incremento de la producción al mantener la línea de producción automatizada, las demoras del proceso son mínimas, no hay agotamiento o desconcentración en las tareas repetitivas, el tiempo de ejecución se disminuye considerablemente según el proceso [3].

1.2 Herramientas de automatización industrial

Con la implementación de métodos numéricos en dispositivos de automatización el resultado es una gama de aplicaciones de rápida expansión y de enfoque especializado en la industria. La tecnología informática, junto con los mecanismos y procesos industriales, pueden ayudar en el diseño, implementación y monitoreo de sistemas de control. Un ejemplo de un sistema de control industrial es el sistema de control y adquisición de datos *SCADA*. La interfaz hombre-máquina *HMI* o interfaces hombre computadora, se suelen utilizar para comunicarse con los controladores lógicos programables (*PLC's*, por sus siglas en ingles) y otros equipos. El personal de servicio se encarga del seguimiento y control del proceso a través de los *HMI*, en donde no solo puede visualizar el estado actual del proceso sino también hacer modificaciones a variables críticas del proceso [3].

1.2.1 Sistemas SCADA

Los sistemas de Supervisión, Control y Adquisición de Datos consisten en una aplicación informática especialmente diseñada para funcionar sobre ordenadores en el control de producción, proporcionando comunicación con los dispositivos de campo (controladores autónomos, autómatas programables PLC, etc.) y controlando el proceso de forma automática desde la pantalla de un ordenador. Además, provee de toda la información que se genera en el proceso productivo a diversos usuarios, tanto del mismo nivel como de otros supervisores dentro de la empresa, control de calidad, supervisión, mantenimiento, etc. [1].

Por otra parte, dichos sistemas cuentan con una segmentación de las tareas y procesos, llamados módulos, cada uno desempeña un rol importante dentro de la supervisión y el control del sistema, funcionando como un engranaje perfectamente alineado.

1.2.2 Módulos de SCADA

Un *SCADA* se compone de bloques o módulos independientes que intervienen en su funcionamiento y permiten las actividades de supervisión, control y adquisición [1].

1. **Configuración:** permite al usuario definir el entorno de trabajo de su *SCADA*, adaptándolo a la aplicación particular que se desea desarrollar.
2. **Interfaz Hombre-Máquina:** proporciona al operador las funciones de control y supervisión de la planta. El proceso se representa mediante sinópticos gráficos almacenados en el ordenador de

proceso y generados desde el editor incorporado en el *SCADA* o importados desde otra aplicación durante la configuración del paquete.

3. **Adquisición:** ejecuta las acciones de mando pre programadas a partir de los valores actuales de variables leídas.
4. **Histórico:** se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.
5. **Comunicación:** se encarga de la transferencia de información entre la planta y la arquitectura *hardware* que soporta el *SCADA*, y entre ésta y el resto de elementos informáticos de gestión.
6. **Seguridad:** es el módulo encargado de mantener el sistema seguro de ataques, de posibles usos indebidos por personal no autorizado y de encriptar toda la información que viaja en la red de datos del *SCADA*.

1.2.3 Citect SCADA

Desarrollado por *Schneider Electric*, Citect *SCADA* es el mayor *SCADA* del mundo, proporcionando soluciones para las necesidades de automatización industrial de sus procesos de producción de minerales y electricidad. Como resultado, su *software* de operación controla 450.000 variables en tiempo real, por lo que un simple segundo de inactividad puede significar la perdida de mucha cantidad de información si se tiene en cuenta que durante las pruebas se controlaron 3 millones de señales digitales en una hora, generando 14 Gb de información total al finalizar las mismas. Para modernizar el sistema de control, se definieron una serie de características, entre ellas que el sistema tendría que ser abierto, para poder facilitar las tareas de mantenimiento y soportar un elevado grado de escalabilidad dadas las posibles ampliaciones que podrían realizarse en un futuro. También se implementó el sistema *Smart Grid*[1], como explican los profesionales de Schneider Electric, incorpora equipos inteligentes en la red eléctrica, desde el punto de consumo en las habitaciones hasta el sistema de transmisión en Alta Tensión, pasando por toda la distribución eléctrica. “El *SCADA* es un elemento de esta cadena que, si bien es diseñado, es el elemento que trabaja los datos que vienen de los equipos y sistemas inteligentes, presentándolos de manera organizada y fácil para el operador del sistema” [4].

Con la investigación del *SCADA* Citect, se comprobó la magnitud que puede alcanzar un sistema de este tipo, cuantas variables se pudieran controlar simultáneamente y cuánta información puede llegar a generar el mismo. Por otra parte, en su paleta de componentes, *Citect SCADA* cuenta con representación de

[1] *Smart Grid* – Red Inteligente

componentes de tipo motores eléctricos, tanto para la industria minera como la eléctrica, las mismas desarrolladas en *softwares* privativos sobre el marco de trabajo *.Net Framework* en el lenguaje C#.

Una vez investigado el sistema, se llega a la conclusión que los conceptos, flujos de acciones y componentes gráficos del Citect *SCADA* no puede ser reutilizados para la solución propuesta, ya que:

- ✓ Utiliza un marco de trabajo privativo.
- ✓ Su marco conceptual no puede ser utilizado libremente.
- ✓ El lenguaje de programación no es el utilizado en el *SCADA* SAINUX 2.0.
- ✓ Para su utilización, se debe comprar la licencia de uso.
- ✓ Es usado sobre sistemas operativos bajo licencias.
- ✓ No utiliza lenguajes *Open Source*.

1.2.4 SCADA SAINUX 2.0

SAINUX integra las características y funcionalidades de alto nivel que permiten la supervisión y control de procesos, utilizando para ello una arquitectura distribuida de módulos para su eficiente funcionamiento y desarrollo. Los módulos o subsistemas funcionales que permiten las actividades de adquisición, supervisión y control de los procesos en el *SCADA* SAINUX 2.0 son los siguientes: comunicación, configuración, seguridad, base de datos históricos, adquisición y *HMI*.

1.3 Interfaz Hombre-Máquina (*HMI*)

Interfaz de usuario asistida por ordenador, actualmente una interfaz de uso, también conocida como interfaz hombre-máquina, forma parte del programa informático que se comunica con el usuario. En ISO 9241-110 (6), el término interfaz de usuario se define como todas las partes de un sistema interactivo (*software* o *hardware*) que proporcionan la información y el control necesarios para que el usuario lleve a cabo una tarea con el sistema interactivo. Muestra las estaciones remotas, componentes, sensores, dispositivos de campo y el sistema de comunicación implicado en el proceso para garantizar su control total [5].

1.3.1 Entorno de configuración del HMI.

Ambiente el cual permite configurar los procesos, gestionar las variables, editar los controladores, comandos, alarmas y demás opciones para la supervisión y el control (Editor gráfico). El editor de configuración además, permite definir el ambiente de trabajo del *SCADA*, adaptando mejor la aplicación al proceso que se desea supervisar, aumentando así mismo las funcionalidades del sistema [5].

1.3.2 Funcionalidades del HMI

Para la supervisión de un proceso industrial, es de vital importancia poseer una interfaz gráfica para la visión real del mismo. Entre las funcionalidades que debe brindar el editor de configuración del *HMI* se encuentran [5]:

1. **Supervisión:** Funcionalidad que junto al monitoreo permite ajustar las condiciones de trabajo del proceso directamente desde la computadora.
2. **Control:** Capacidad de aplicar algoritmos que ajustan los valores del proceso y mantenerlos dentro de ciertos límites.
3. **Monitoreo:** Habilidad de mostrar los datos del proceso en tiempo real.
4. **Históricos:** Capacidad de muestrear y almacenar en un archivo datos del proceso a una determinada frecuencia. Es una poderosa herramienta para la optimización y corrección de procesos.
5. **Alarmas:** Es la capacidad de reconocer eventos excepcionales dentro del proceso y reportarlos. Las alarmas son reportadas a partir de límites de control preestablecidos.

1.3.3 Editor gráfico

El editor gráfico es la interfaz o mecanismo que permite al operador del *SCADA* cree sinópticos de los procesos industriales. Dichos sinópticos se representan mediante objetos y premisas básicas predefinidas que se pueden combinar, agrupar, importar, exportar y transformar permitiendo que el componente gráfico sea el que permita la creación y edición de los objetos y despliegues presentes en los escenarios productivos. Para ello se emplean un conjunto de herramientas entre las que se encuentran [5]:

- ✓ **Barra de herramientas:** Contiene los botones que permiten realizar rápidamente las acciones más frecuentes en el editor gráfico, entre las que se encuentran copiar, cortar, pegar, deshacer, rehacer, enviar al fondo, traer al frente, acercar y alejar un objeto gráfico.
- ✓ **Barra de menú:** Contiene las acciones para el diseño gráfico.
- ✓ **Inspector de propiedades:** Los elementos gráficos y recursos en el sistema tienen propiedades que se deben modificar y explorar por los operadores de la aplicación. El inspector de propiedades es la vista que permite ver las propiedades.
- ✓ **Paleta de objetos:** Es la herramienta que contiene los objetos gráficos. Los objetos gráficos orientados a objetos, también llamados gráficos estructurados, son gráficos de ordenador basados en el uso de elementos de construcción, como líneas, curvas, círculos y rectángulos que presentan

un conjunto de propiedades que pueden ser editadas y asociadas a variables del *SCADA*. Por medio de los objetos gráficos se crean animaciones en función de los valores de las variables asociadas.

- ✓ **Área de edición:** Es el área de la aplicación donde se crean los despliegues y los reportes, seleccionando los elementos gráficos que se brindan en la paleta de objetos. Permite acceder a las propiedades de los elementos gráficos para construir las pantallas que se muestran en tiempo de ejecución del sistema.

1.3.4 Ambiente de ejecución del HMI

El ambiente de ejecución es el encargado de hacer visuales las animaciones y los objetos definidos en el ambiente de configuración, mostrando en tiempo real lo ocurrido en el proceso. Al ser el *HMI* el módulo que se encarga de proporcionar el control total del proceso directamente, es el que envía las órdenes a las estaciones remotas, quienes reciben los valores de las variables e interactúan con la mayoría de los operadores encargados de supervisar el proceso directamente. Así se facilitan las acciones primarias como la generación de los reportes, el análisis de las variables y sus valores, visualización de la tendencia de los marcadores, la impresión de los reportes y el acceso a las alarmas que se disparan para su inmediata supervisión [5].

1.4 Motores Eléctricos para la industria

Los motores eléctricos son máquinas eléctricas rotatorias que transforman la energía eléctrica en energía mecánica. En la industria se emplean para accionar máquinas-herramienta, bombas, montacargas, ventiladores, extractores, elevadores, grúas eléctricas, etc. [6].

1.4.1 Tipos de los motores eléctricos

Los motores eléctricos se clasifican en dos grupos; los motores de corriente directa o continua (DC por sus siglas en inglés), y de corriente alterna (AC por sus siglas en inglés).

1.4.2 Motores eléctricos de corriente directa

Un motor de corriente directa funciona con carga al realizar un determinado trabajo (halando, empujando objetos o soportando cualquier resistencia externa o carga) que lo obliga a absorber energía mecánica. Pueden clasificarse en monofásicos o trifásicos, obtienen su clasificación según la forma de conexión de las bobinas inductoras e inducidas entre sí [6].

- ✓ Motor de excitación independiente.
- ✓ Motor serie.

- ✓ Motor de derivación o motor shunt.
- ✓ Motor compoud.

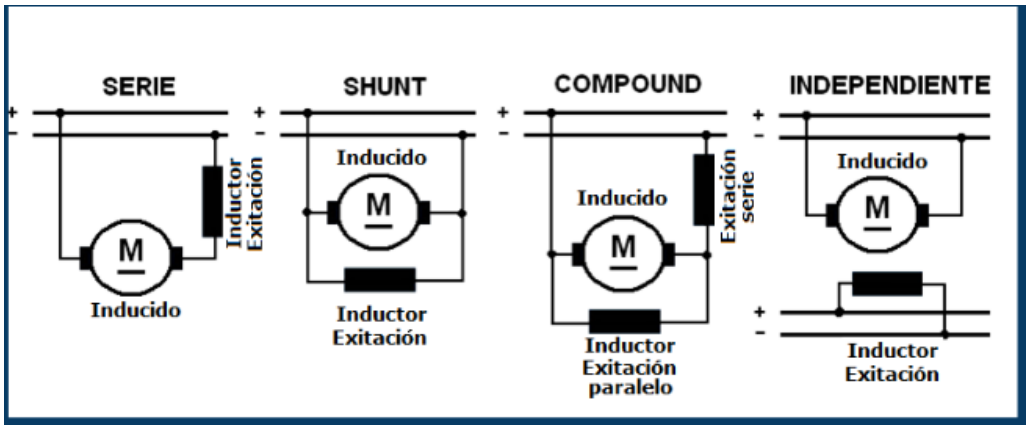


Fig. 1 Clasificación de motores de corriente directa

1.4.3 Motores eléctricos de corriente alterna

Por el fácil manejo de transmisión, distribución y transformación de la corriente alterna, se ha constituido en la corriente con más uso en la sociedad moderna. Es por ello que los motores de corriente alterna, son los normales en el uso cotidiano de las personas y las empresas, y con el desarrollo tecnológico se ha conseguido un rendimiento altísimo que hace que más del 90 % de los motores instalados sean de corriente alterna [6].

Se clasifican en dos grupos, síncronos y asíncronos.

Síncronos: son motores de corriente alterna en el que la rotación del eje está sincronizada con la frecuencia de la corriente de alimentación.

Asíncronos: motores de corriente alterna en el que la rotación del eje es menor que la frecuencia del campo de sincronismo.

1.5 Componentes definidos

El estándar seguido para la utilización de los símbolos eléctricos está definido por la Comisión Electrotécnica Internacional (*IEC*, por sus siglas en inglés), dicha organización se encarga de la normalización en los campos: eléctrico, electrónico y tecnologías relacionadas. Más concretamente los gráficos de la simbología eléctrica se rige en la norma europea EN 60617 aprobada por el Comité Europeo de Normalización Electrotécnica (CENELEC) bajo la Norma Internacional *IEC* 61082 [2].

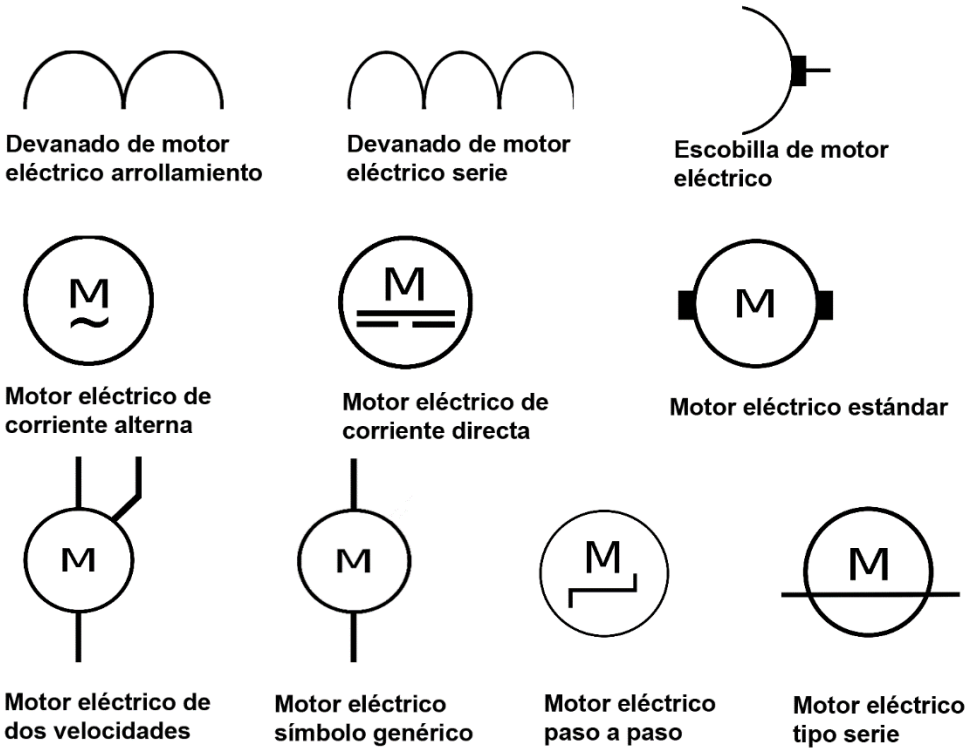


Fig. 2 Simbología de motores definidos

1.6 Herramientas y tecnologías para el desarrollo

Las herramientas y tecnologías escogidas para el desarrollo de un proyecto deben ser seleccionadas cuidadosamente, ya que pueden suponer el fracaso de éste o pueden aumentar su complejidad, para lo cual se deben conocer cuáles son las distintas alternativas y las necesidades del proyecto. La selección de dichas herramientas y tecnologías está fundamentada para su utilización en el proyecto SAINUX 2.0.

1.6.1 Inkscape

Editor de gráficos vectoriales de código abierto, similar a programas como *Adobe Illustrator, Corel Draw, FreeHand* o *Xara* X. Lo que lo hace único es que usa como formato nativo el Gráfico de Vector Escalable (SVG, por sus siglas en inglés), un estándar abierto de W3C basado en XML, soportando formas, trazos, texto, marcadores, clones, mezclas de canales alfa, transformaciones, gradientes, patrones y agrupamientos [7].

Como objetivo tiene proporcionar a los usuarios una herramienta libre de código abierto de elaboración de gráficos en formato vectorial escalable (SVG) que cumpla completamente con los estándares XML, SVG y

CSS2. Se encuentra desarrollado principalmente para el sistema operativo GNU/Linux, pero es una herramienta multiplataforma y funciona en Windows, Mac OS X, y otros sistemas derivados de Unix.

1.6.2 Gráficos Vectoriales Escalables

El formato SVG especifica un gráfico vectorial con gran facilidad para escalar (Scalable Vector Graphics). Los ficheros SVG se definen en XML y permiten usar formas gráficas, mapas de bits o texto. Al mismo tiempo pueden ser estáticos o dinámicos. Al igual que el Adobe Flash sus características vectoriales le posibilitan que el escalado de las imágenes sea óptimo tanto para aumentar la imagen como para disminuirla. Por otro lado, cuando escalamos una imagen de mapa de bits (PNG, JPG, BMP) la imagen se distorsiona y no ofrece una calidad apropiada. Los gráficos vectoriales pueden ser transformados (estirar, rotar, mover, distorsionar) de una manera más sencilla y con menos requerimientos de memoria en el equipo [8].

1.6.3 Lenguaje de programación

Un lenguaje de programación es un conjunto de símbolos, reglas sintácticas y semánticas que definen su estructura, el significado de sus elementos y sus expresiones. Es utilizado para controlar el comportamiento físico y lógico de un computador, programar controladores de dispositivos y otros *softwares* que se basen en la manipulación directa de *hardware*, bajo restricciones de tiempo real. El lenguaje en el cual se implementará la solución será C++, el cual fue diseñado para que cada característica del lenguaje se pudiera utilizar en virtud de limitaciones severas de tiempo y espacio, utilizando vectores como principal recurso para acceder y modificar la memoria del computador. Por otra parte, dicho lenguaje de programación ofrece flexibilidad lo que permite programar con múltiples estilos, uno de los más empleados es el estructurado "no llevado al extremo".

1.6.4 Marco de trabajo

Un marco de trabajo o *framework* es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación. Es una definición muy genérica, pero también puede serlo un *framework*. En el otro extremo, otros *frameworks* pueden llegar al detalle de definir los nombres de ficheros, su estructura, las convenciones de programación, etc. [9].

Qt es un f*ramework* multiplataforma orientado a objetos, se utiliza para desarrollar *software* que utilicen interfaz gráfica de usuario, así como también diferentes tipos de herramientas para la línea de comandos y consolas para servidores que no necesitan una interfaz gráfica de usuario. Incluye clases, bibliotecas y herramientas para la producción de aplicaciones usando el lenguaje C++ de forma nativa. Funciona en las

principales plataformas y tiene un amplio apoyo operando en varios sistemas como Unix (Linux, MacOS X, Solaris) o incluso toda la familia de Windows. El principal motivo por el que Qt se ha hecho tan popular es por sus métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos y una multitud de otros métodos para el manejo de ficheros, además de estructuras de datos tradicionales [9].

1.6.6 Entorno Integrado de Desarrollo (IDE)

Para el desarrollo de este producto se requiere de un entorno de desarrollo integrado (*IDE*, por sus siglas en inglés), el cual puede denominarse como un entorno de programación que consiste en un editor de código y un compilador [10].

El *IDE* seleccionado para el desarrollo de la aplicación es Qt Creator en su versión 4.9.2. Qt Creator es un *IDE* multiplataforma para el desarrollo de aplicaciones que pueden o no tener interfaz gráfica. Este se centra en proporcionar características que ayudan a los nuevos usuarios del *IDE* a aprender y comenzar a desarrollar rápidamente. Tiene la disponibilidad del código fuente, la excelente documentación organizada que provee en el Qt *Assistant* y un editor para el diseño de formularios denominado Qt *Designer*. Qt *Creator* cuenta con [10]:

- ✓ Un editor de código con soporte para C++.
- ✓ Herramientas para la rápida navegación por el código.
- ✓ Resaltado de sintaxis y auto-completado de código.
- ✓ Control estático de código y estilo a medida.
- ✓ Soporte para refactorización de código.
- ✓ Paréntesis coincidentes y modos de selección.

1.6.7 Herramienta CASE

Las herramientas de Ingeniería de *Software* Asistida por Computadora (*CASE*, por sus siglas en inglés), son aplicaciones informáticas utilizadas en el proceso de desarrollo de *software* en tareas con el objetivo de realizar un diseño del proyecto, cálculo de costos, implementación de parte del código automáticamente a partir del diseño, compilación automática, documentación o detección de errores entre otras. *Visual Paradigm* es una herramienta *CASE* profesional, que soporta el ciclo de vida completo del desarrollo de *software*, ya sea el análisis y diseño orientados a objetos, construcción, pruebas y despliegue [12]. Este *software* fue el utilizado para el modelado de la solución, cuyas características principales son las que a continuación se enuncian:

- ✓ Soporte de *UML*.
- ✓ Ingeniería inversa – Código a modelo, código a diagrama.
- ✓ Generación de código – Modelo a código, diagrama a código.
- ✓ Generación de bases de datos -Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- ✓ Ingeniería inversa de bases de datos -Desde Sistemas Gestores de Bases de Datos existentes a diagramas de Entidad-Relación.
- ✓ Editor de figuras.

1.6.8 Lenguaje de Modelado Unificado (UML)

Para el desarrollo del presente trabajo se utiliza el Lenguaje Unificado de Modelado (*UML*, por sus siglas en inglés), utilizado para especificar o describir métodos y procesos. *UML* se utiliza en su versión 2.0, para visualizar, especificar, construir y documentar el sistema. Ofrece un estándar para describir un modelo del sistema, incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables. El lenguaje de modelado es una notación gráfica que indica los pasos que se deben seguir para confeccionar un diseño [11].

1.7 Metodologías de desarrollo

Desarrollar un buen *software* depende de un sinnúmero de actividades y etapas, donde se debe elegir la mejor metodología para un equipo en un determinado proyecto, lo cual es trascendental para el éxito del producto. La UCI cuenta con centros productivos en su gran mayoría pertenecientes a las facultades que conforman la universidad. Cada uno de estos centros se dedica al desarrollo de *software* y/o servicios asociados a un dominio de aplicación bien definido. Esta diversidad de centros y proyectos hace que la actividad productiva en la UCI sea cada vez más amplia, y trae consigo la heterogeneidad en el proceso de desarrollo de *software*.

1.7.1 Metodología AUP-UCI

Para el desarrollo de la presente investigación se utiliza como metodología de desarrollo *AUP*-UCI, ya que genera los artefactos que permiten obtener la documentación necesaria para una mejor comprensión de la herramienta a desarrollar.

El Proceso Unificado Ágil (*AUP*, por sus siglas en inglés) es una versión simplificada del Proceso Racional Unificado (*RUP*, por sus siglas en inglés). Este describe de una manera simple y fácil de entender la forma

de desarrollar aplicaciones informáticas de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en *RUP*.

Al no existir una metodología de *software* universal, ya que toda metodología debe ser adaptada a las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigiéndose así que el proceso sea configurable. Se decide hacer una variación de la metodología *AUP*, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. Una metodología de desarrollo de *software* tiene entre sus objetivos aumentar la calidad del *software* que se produce, de ahí la importancia de aplicar buenas prácticas, para ello nos apoyaremos en el Modelo *CMMI-DEV* v1.3. El cual constituye una guía para aplicar las mejores prácticas en una entidad desarrolladora. Estas prácticas se centran en el desarrollo de productos y servicios de calidad [13].

La variación de la metodología *AUP*, denominada *AUP*-UCI se adapta al ciclo de vida definido para la actividad productiva de la UCI y es utilizada por el CEDIN para el proceso de desarrollo de *software*. El uso de esta técnica de modelado ágil permite encapsular los requisitos funcionales en Historias de Usuario (HU), descripción de casos de uso del sistema y la descripción de requisitos por procesos [13].

1.7.2 Descripción de las fases

De las 4 fases que propone *AUP* (Inicio, Elaboración, Construcción, Transición) se decide mediante el estudio de los métodos ágiles para el ciclo de vida de los proyectos de la UCI mantener la fase de Inicio, pero modificando el objetivo de la misma, se unifican las restantes 3 fases de *AUP* en una sola, a la que llamaremos Ejecución y se agrega la fase de Cierre. Para una mayor comprensión se muestra la siguiente tabla.

Tabla 1 Comparativa de las fases *AUP* y *AUP*-UCI

Fases *AUP*	Fases variación *AUP*-UCI	Objetivos de las fases (Variación *AUP*-UCI)
Inicio	Inicio	Durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.
Elaboración		

Construcción Transición	Ejecución	En esta fase se ejecutan las actividades requeridas para desarrollar el *software*, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.
	Cierre	En esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto.

1.8 Conclusiones parciales

La elaboración del capítulo, permitió un estudio profundo de los sistemas *SCADA*, facilitándose la confección del estudio del estado del arte para un mayor entendimiento del funcionamiento de dichos sistemas, lo que permitió arribar a las siguientes conclusiones:

- ✓ La confección del marco teórico brindó un acercamiento a los principales conceptos, técnicas y métodos a utilizar durante el resto de la investigación.
- ✓ Permitió profundizar los aspectos teóricos de sistemas *SCADA*, sus interfaces hombre-máquina y el análisis de las formas de representación de la información.
- ✓ Propició una base de conocimientos útil para el desarrollo de la investigación.
- ✓ Facilitó el estudio de técnicas y herramientas, consolidando las propuestas tecnológicas necesarias para la realización de los componentes gráficos para el *SCADA* SAINUX 2.0, lográndose así, una fundamentación técnica que justifica la solución escogida.

Capítulo II. Descripción de la solución.

2.1 Introducción

En este capítulo se describen las actividades desarrolladas durante el proceso de descripción de la solución. Se define la lista de requisitos funcionales y no funcionales que se deben tener en cuenta para la elaboración de los componentes gráficos. Se analizan los conceptos fundamentales del modelo de dominio necesarios para comprender el problema planteado y se describe el diagrama de clases, con los métodos utilizados en el sistema.

2.2 Modelo del dominio

Un modelo de dominio asocia las entidades o conceptos que se manejan en el entorno en el que trabaja el sistema mediante un diagrama de clases *UML* [11].

A continuación, se presenta el modelo del dominio que corresponde a la solución, así como la descripción de los conceptos y objetos involucrados.

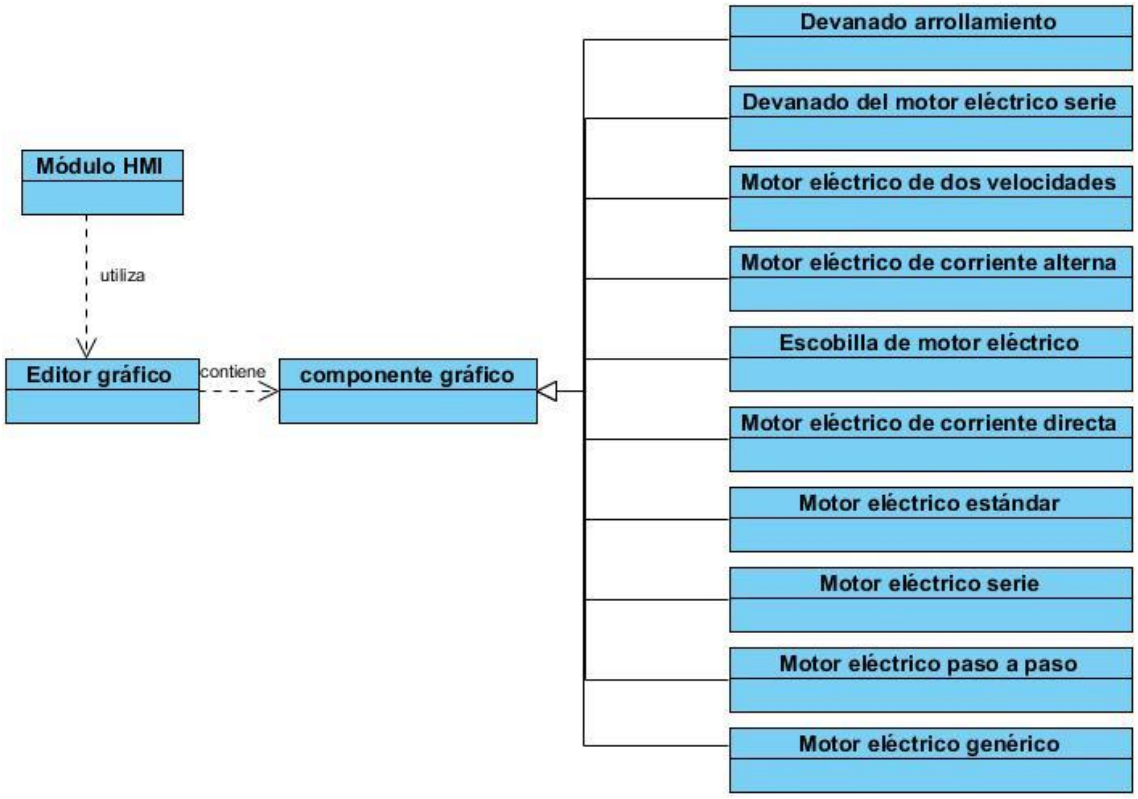


Fig. 3 Modelo del dominio

- ✓ **Módulo *HMI*:** Módulo encargado de representar la interfaz del editor gráfico al usuario.
- ✓ **Editor gráfico:** Interfaz gráfica donde se muestra la paleta de componentes disponibles para la configuración.
- ✓ **Componente gráfico:** Representación abstracta de un dispositivo del proceso que se supervisa.
- ✓ **Devanado del motor eléctrico arrollamiento:** Componente encargado de hacer girar el motor en su sentido de rotación, así mismo, el arrollamiento es un conjunto de cilindros metálicos magnetizados incorporados en el devanado para garantizar la polaridad magnética.
- ✓ **Devanado del motor eléctrico serie:** Componente que garantiza un único sentido de giro a un motor eléctrico, este componente solo es aplicado a motores de corriente directa.
- ✓ **Motor eléctrico de dos velocidades:** Es un tipo de motor eléctrico donde tiene dos velocidades de excitación, baja entre 500 – 1000 revoluciones, y alta entre 1500 – 3000 revoluciones.
- ✓ **Motor eléctrico de corriente alterna:** Motores síncronos. Los motores síncronos son motores de corriente alterna en el que la rotación del eje está sincronizada con la frecuencia de la corriente de alimentación.
- ✓ **Escobilla de motor eléctrico:** Componente encargado de establecer una conexión eléctrica entre la parte fija y la parte rotatoria del motor eléctrico.
- ✓ **Motor eléctrico de corriente directa:** Motor eléctrico que funciona con carga de baterías o generadores de bajo voltaje.
- ✓ **Motor eléctrico estándar:** Tipo de motor eléctrico que dada su bajo consumo eléctrico puede funcionar con corriente directa o alterna, generando revoluciones inferiores a los 500 *rpm*[2].
- ✓ **Motor eléctrico serie:** El motor serie o motores de excitación en serie, es un tipo de motor eléctrico de corriente continua en el cual el inducido y el devanado inductor o de excitación van conectados en serie, el voltaje aplicado es constante, mientras que el campo de excitación aumenta con la carga.
- ✓ **Motor eléctrico paso a paso:** Conocido también como motor de pasos, es un dispositivo electromecánico que convierte una serie de impulsos eléctricos en desplazamientos angulares discretos, lo que significa que es capaz de girar una cantidad de grados (paso o medio paso) dependiendo de sus entradas de control.
- ✓ **Motor eléctrico genérico:** Forma gráfica de representación de un motor eléctrico, en la cual no se define su tipo de corriente, ni revoluciones por minuto.

[2] RPM: Revoluciones Por Minuto

2.3 Captura de requisitos

Un requisito es una condición o capacidad que necesita el usuario para resolver un problema o conseguir un objetivo determinado. También se aplica a las condiciones que debe cumplir o poseer un sistema o uno de sus componentes para satisfacer un contrato, una norma o una especificación; pueden ser funcionales o no funcionales [14].

2.3.1 Requisitos funcionales (RF)

Los requisitos funcionales son una definición de los servicios que el sistema debe proporcionar, cómo debe reaccionar a una entrada particular y cómo se debe comportar ante situaciones particulares [14].

A continuación, se muestran los requisitos a implementar en la propuesta de solución.

RF1: Mostrar los componentes en la Paleta de componentes.

RF2: Arrastrar el componente hacia el despliegue.

RF3: Visualizar el componente en el despliegue.

RF4: Eliminar el componente motor eléctrico en el despliegue.

RF5: Modificar el tamaño del componente motor eléctrico.

RF6: Configurar las propiedades del componente.

RF7: Configurar color de las líneas de contorno del componente.

2.3.2 Requisitos no funcionales (RNF)

Los requisitos no funcionales son restricciones que afectan a los servicios o funciones del sistema, tales como restricciones de tiempo, sobre el proceso de desarrollo, estándares. Definen propiedades emergentes del sistema, tales como el tiempo de respuesta, la fiabilidad, entre otras [14].

RNF1. Software:

- ✓ El sistema debe funcionar en el Sistema Operativo GNU-Linux.

RNF2. *Hardware*:

- ✓ Debe ser ejecutado en computadoras que tengan como requerimientos mínimos los siguientes:
  - Microprocesador: dual Core 1.5 GHz.
  - RAM: 2Gb

RNF3. Restricciones en el diseño y la implementación:

- ✓ Se implementará utilizando el lenguaje de programación C++.
- ✓ Se empleará el *framework* Qt, en su versión 5.0.
- ✓ Se utilizará como *IDE* de desarrollo Qt Creator, en su versión 4.9.2.

2.3.3 Historias de usuario

Los modelos ágiles utilizan historias de usuario para captar las necesidades de los clientes en un proyecto de *software*. Una historia de usuario, es una descripción en primera persona de una acción que el usuario efectúa en un sistema [15].

A continuación, se presentan las historias de usuario que forman parte de la propuesta de solución.

Tabla 2 HU 1: Visualizar los componentes en la paleta de componentes

Historia de usuario	
Número: HU1	**Nombre del requisito:** RF1: Visualizar los componentes en la paleta de componentes.
Programador: Alejandro Cabo Marchena	**Iteración asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: - Problemas eléctricos - Problemas técnicos	**Tiempo real:** 1 semana
Descripción: El operador podrá observar los componentes disponibles en la paleta de componentes para su inserción al Despliegue.	
Prototipo de interfaz:	

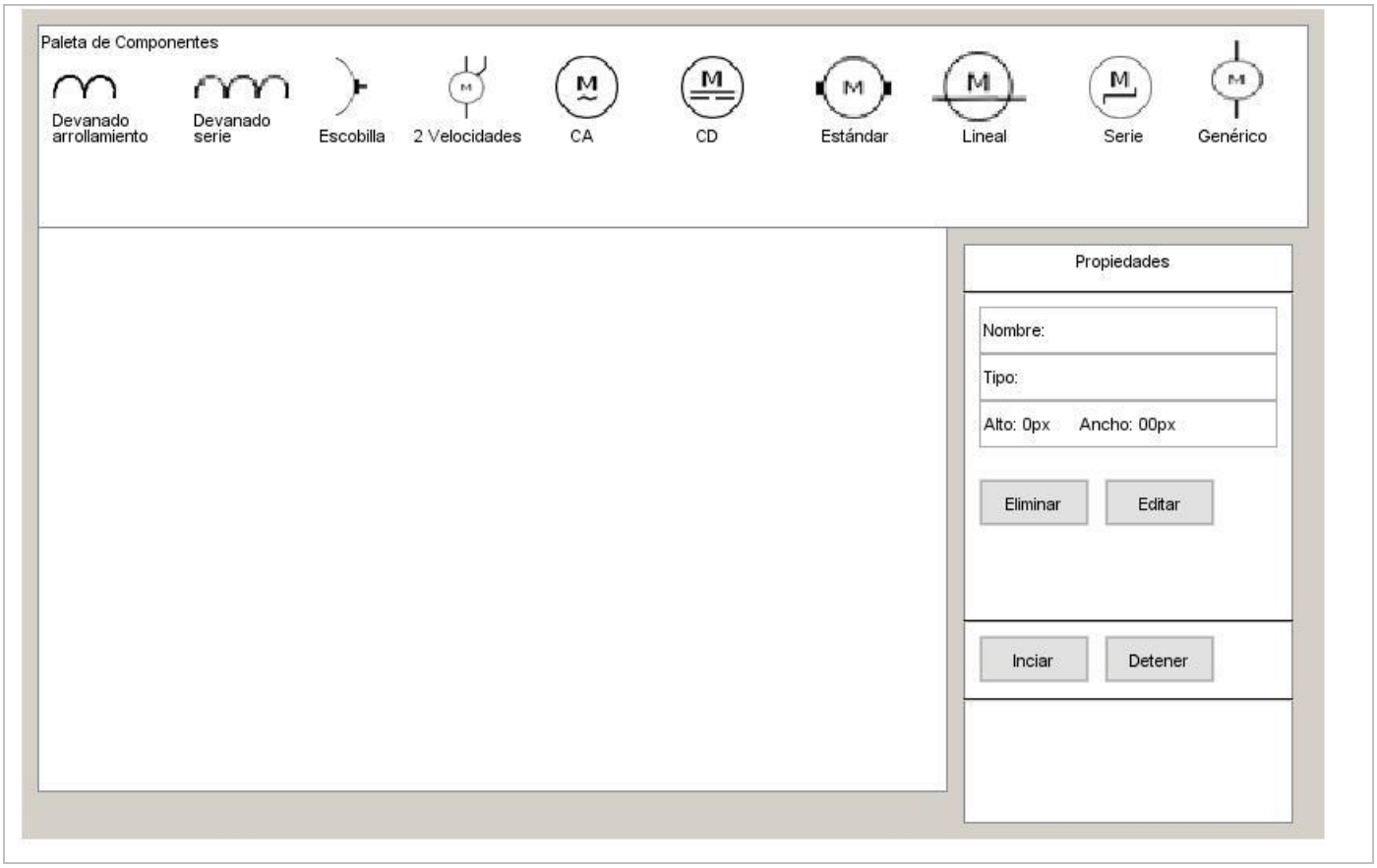


Tabla 3 HU2: Arrastrar el componente hacia el despliegue.

Historia de usuario	
Número: HU2	**Nombre del requisito:** RF2: Arrastrar el componente hacia el despliegue.
Programador: Alejandro Cabo Marchena	**Iteración asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: - Problemas eléctricos - Problemas técnicos	**Tiempo real:** 1 semana
Descripción: El operador podrá adicionar el componente gráfico tipo motor eléctrico al despliegue del editor gráfico. Para dicha operación el operador deberá seleccionar el motor eléctrico de la paleta de componentes y hacerle *drop* hacia dentro del despliegue.	
Prototipo de interfaz:	

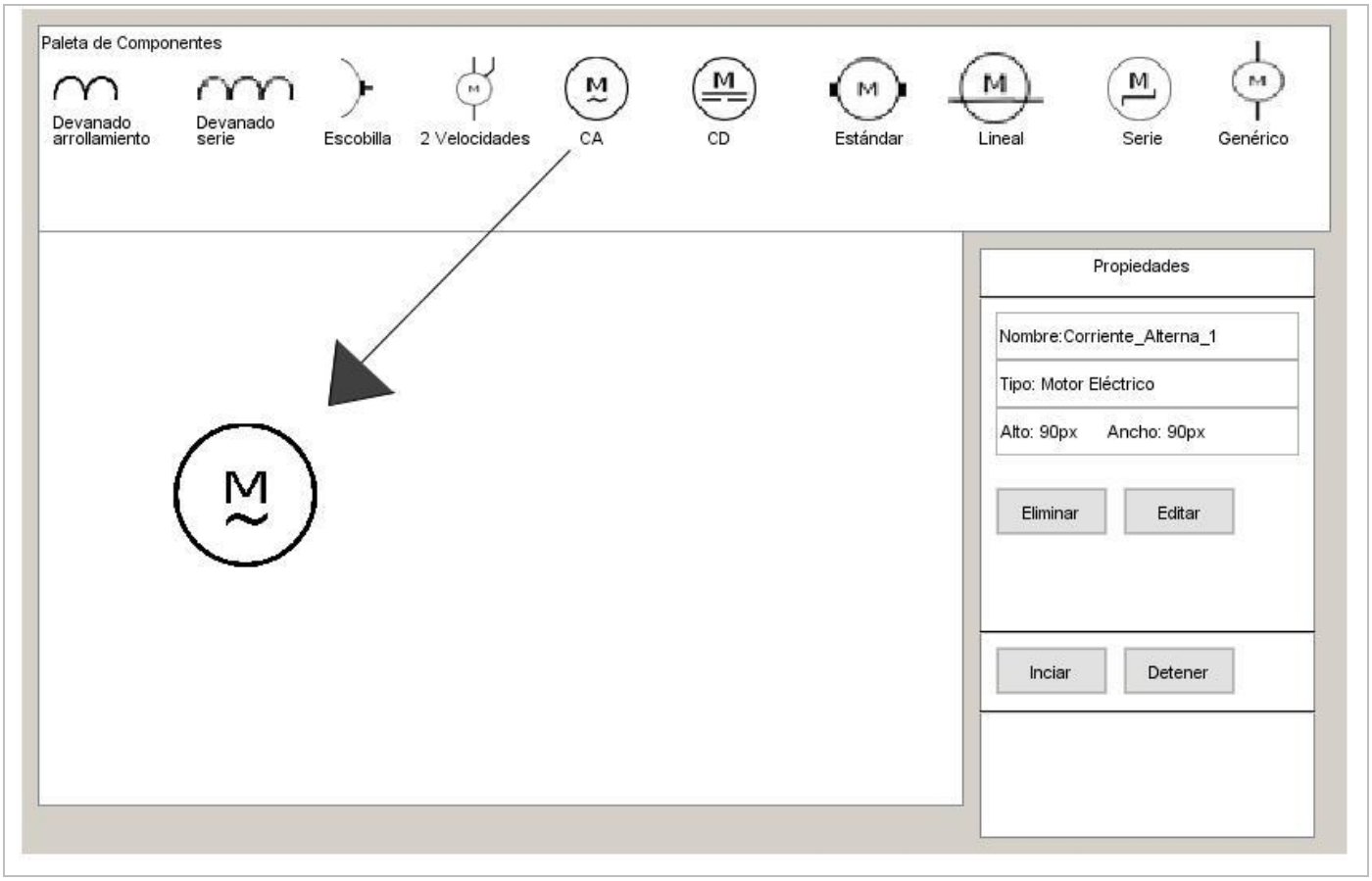


Tabla 4 HU3: Visualizar el componente gráfico en el Runtime.

Historia de usuario	
Número: HU3	**Nombre del requisito:** RF3: Visualizar el componente gráfico en el despliegue.
Programador: Alejandro Cabo Marchena	**Iteración asignada:** 3
Prioridad: Alta	**Tiempo Estimado:** 3 semana
Riesgo en Desarrollo: - Problemas eléctricos - Problemas técnicos	**Tiempo real:** 1 semana
Descripción: El operador podrá visualizar el componente gráfico tipo motor eléctrico en el *Runtime*. Para esto una vez configurado los componentes, el operario debe correr el *Runtime* o Visualizador Gráfico, donde se visualizará la red eléctrica con los motores.	
Prototipo de interfaz:	

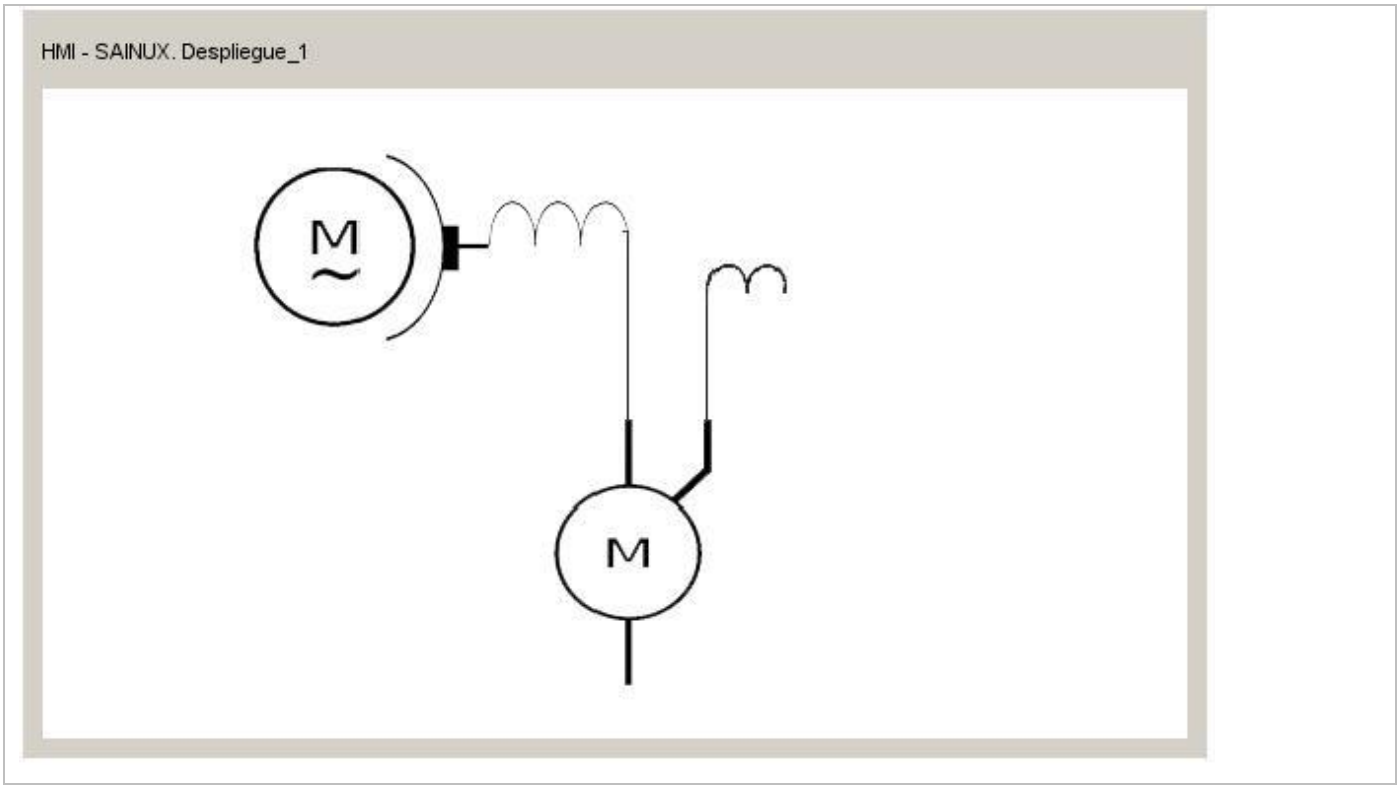


Tabla 5 HU4: Eliminar el componente del despliegue

Historia de usuario	
Número: HU4	**Nombre del requisito:** RF4: Eliminar el componente del despliegue.
Programador: Alejandro Cabo Marchena	**Iteración asignada:** 2
Prioridad: Alta	**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: - Problemas eléctricos - Problemas técnicos	**Tiempo real**: 1 semana
Descripción: El operador podrá eliminar el componente gráfico tipo motor eléctrico en el despliegue del editor gráfico. Para dicha operación el operador deberá seleccionar el motor eléctrico en el despliegue y presionar le tecla *Supr*, una vez hecha la operación aparecerá una ventana emergente de confirmación de eliminado.	
Prototipo de interfaz:	

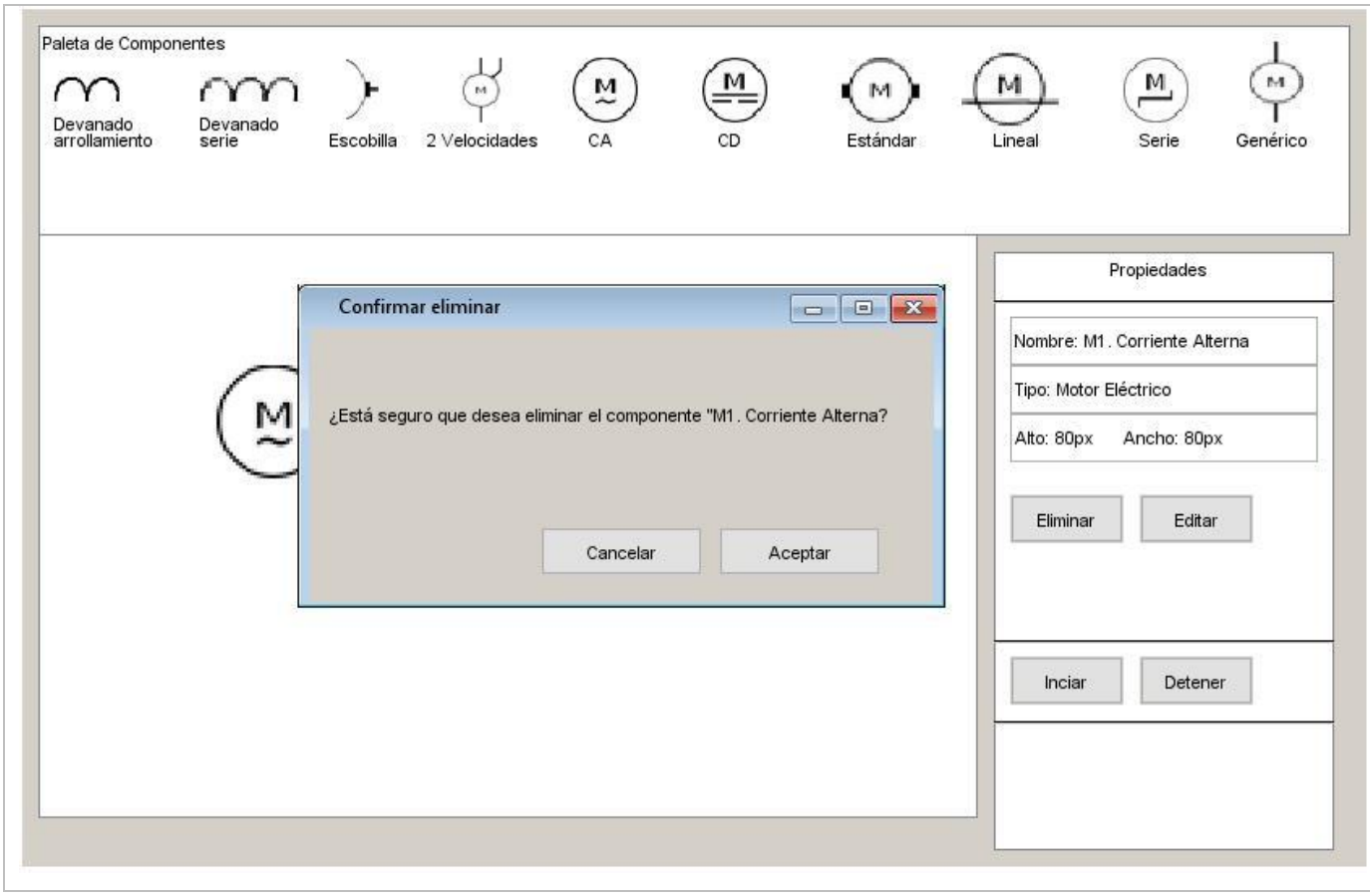


Tabla 6 HU5: Modificar el tamaño del componente en el despliegue

Historia de usuario	
Número: HU5	**Nombre del requisito:** RF5: Modificar el tamaño del componente en el despliegue.
Programador: Alejandro Cabo Marchena	**Iteración asignada:** 3
Prioridad: Alta	**Tiempo Estimado:** 1 semana
Riesgo en Desarrollo: - Problemas eléctricos - Problemas técnicos	**Tiempo real:** 1 semana
Descripción: El operador podrá modificar las propiedades del componente gráfico tipo motor eléctrico en el despliegue del editor gráfico. Para dicha operación el operador deberá seleccionar el motor eléctrico en el despliegue, aparecerán puntos de edición alrededor del mismo y se debe estirar y contraer con el puntero hasta obtener el tamaño deseado, de lo contrario en el visor de propiedades modificar el valor al deseado. El valor por defecto al añadir el componente es de 90 píxeles.	

Prototipo de interfaz:

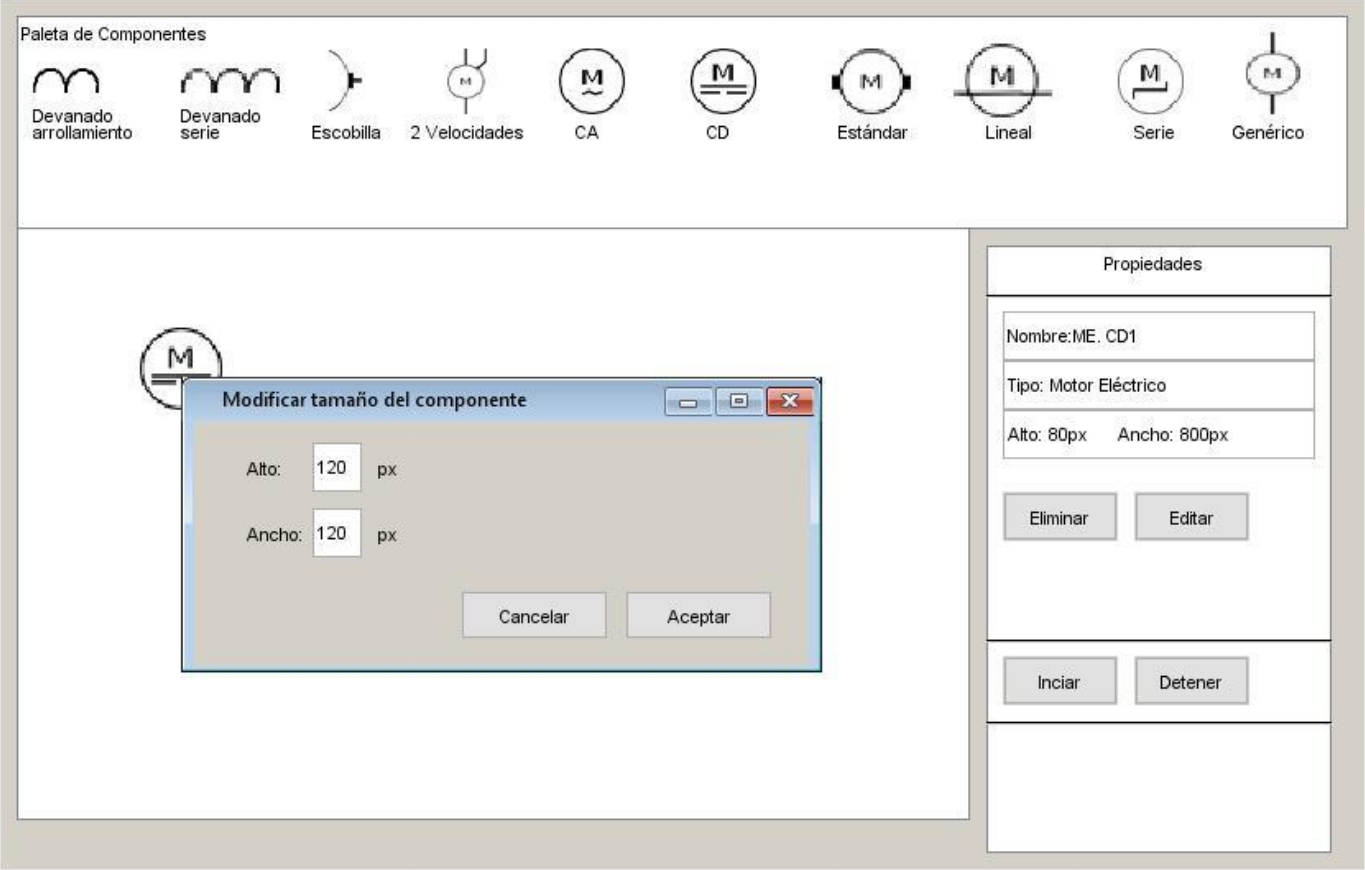


Tabla 7 HU6: Configurar las propiedades del componente

Historia de usuario	
Número: HU6	**Nombre del requisito:** RF6: Configurar propiedades del componente en el despliegue
Programador: Alejandro Cabo Marchena	**Iteración asignada:** 2
Prioridad: Alta	**Tiempo Estimado:** 1 semana
Riesgo en Desarrollo: - Problemas eléctricos - Problemas técnicos	**Tiempo real:** 1 semana
Descripción: El operador podrá asignarle propiedades al componente gráfico tipo motor eléctrico. Para dicha operación el operador debe seleccionar el componente en el despliegue, y en el inspector de propiedades modificar los valores tipo nombre y descripción en el visor de propiedades.	

Prototipo de interfaz:

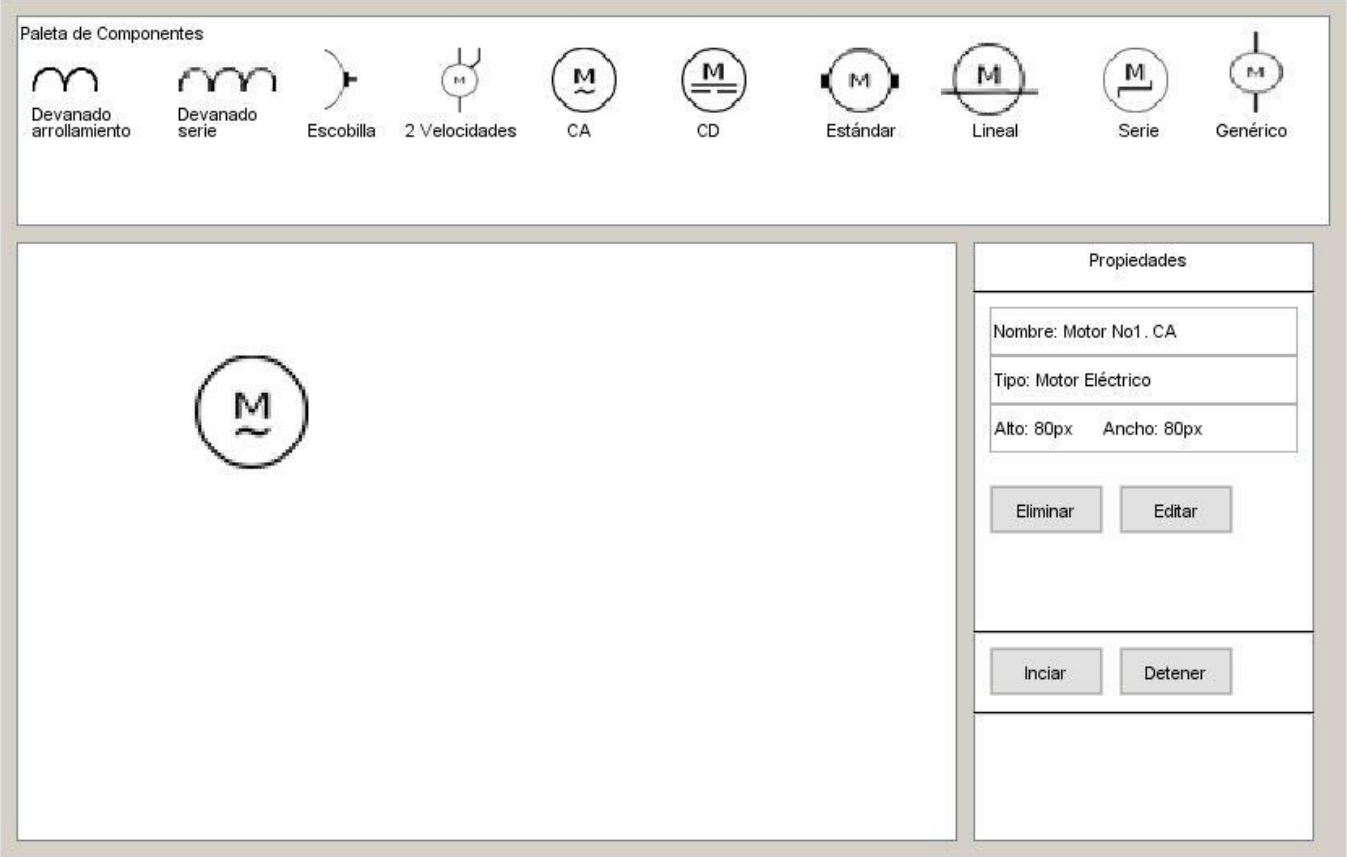
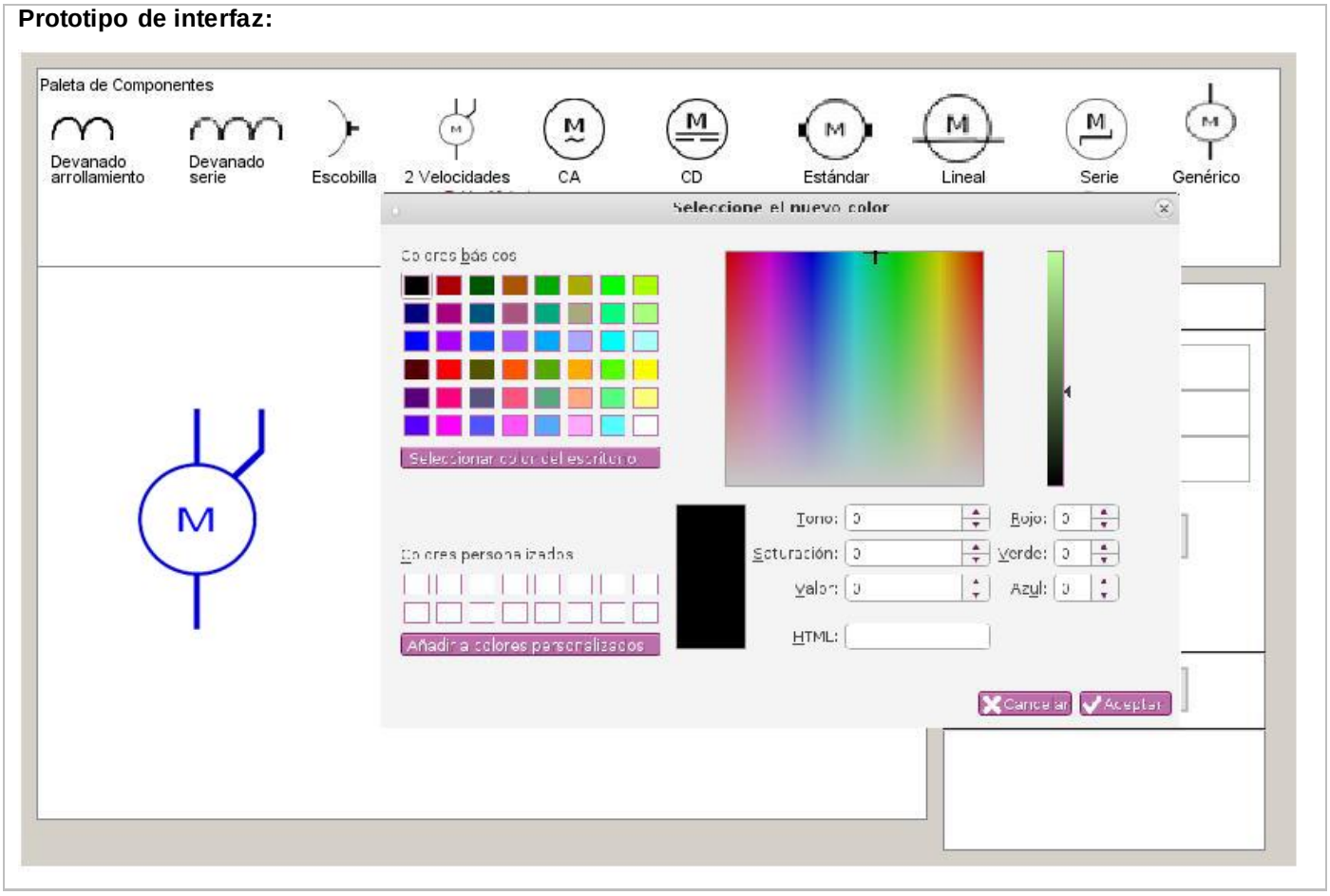


Tabla 8 HU7: Cambiar color de las líneas de contorno del componente.

Historia de usuario	
Número: HU7	**Nombre del requisito:** RF7: Cambiar color de las líneas de contorno del componente.
Programador: Alejandro Cabo Marchena	**Iteración asignada:** 3
Prioridad: Alta	**Tiempo Estimado:** 3 semanas
Riesgo en Desarrollo: - Problemas eléctricos - Problemas técnicos	**Tiempo real:** 1 semana
Descripción: El operador podrá configurar el color de las líneas de contorno del componente eléctrico, para esto, se abre el editor de propiedades, selecciona "*border line color*" y se escoge el color deseado o puede entrar manualmente el valor del color requerido en la paleta de colores.	

Prototipo de interfaz:



2.4 Descripción de la solución

Los componentes propuestos serán incorporados a la paleta de visualización gráfica existentes actualmente en el *HMI* del *SCADA* SAINUX 2.0, permitiendo representar motores eléctricos en el *Runtime* del sistema y ha de cumplir los requisitos descritos en el epígrafe anterior.

2.5 Diagrama de Clases

Un diagrama de clases es utilizado para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso y de agregación, ya que una clase es una descripción de conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica; mostrando un conjunto de elementos que son estáticos, como las clases y tipos junto con sus contenidos y relaciones. Cuando se crea un diagrama de clases, se está modelando una parte de los elementos y relaciones que configuran la vista de diseño del sistema [14].

A continuación, se muestra el diagrama de clases del diseño teniendo en cuenta las entidades, sus atributos y relaciones.

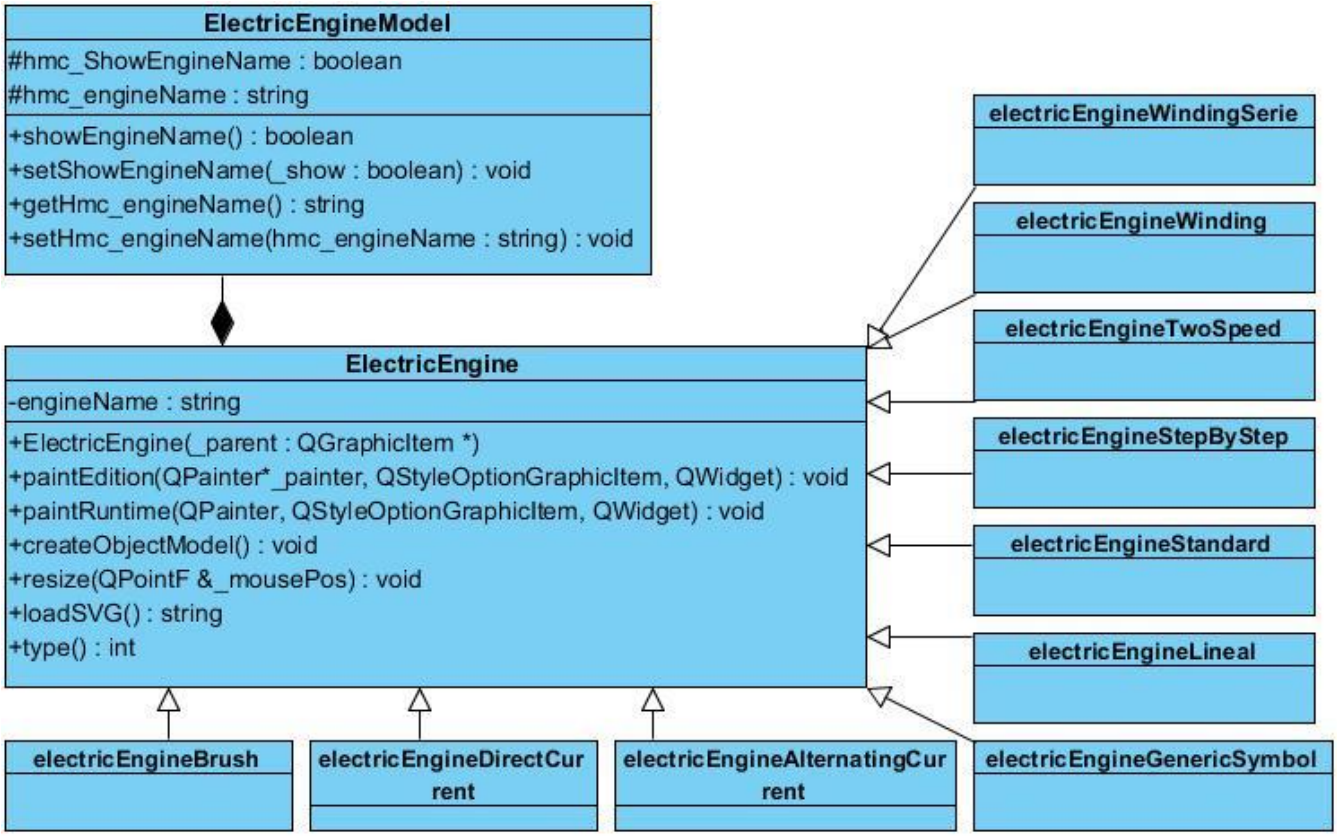


Fig. 4 Diagrama de Clases del Sistema

2.6 Arquitectura del sistema

Una arquitectura de *software* consiste en un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del *software* para un sistema de información [15].

La arquitectura utilizada para el desarrollo de la solución propuesta es el Modelo Vista Controlador, dicho paradigma es la arquitectura base de *Qt framework*, utilizado en el desarrollo del sistema.

2.6.1 Modelo Vista Controlador

El patrón Modelo-Vista-Controlador (MVC) surge con el objetivo de reducir el esfuerzo de programación, necesario en la implementación de sistemas múltiples y sincronizados de los mismos datos, a partir de estandarizar el diseño de las aplicaciones. El patrón MVC es un paradigma que divide las partes que conforman una aplicación en el Modelo, las Vistas y los Controladores, permitiendo la implementación por

separado de cada elemento, garantizando así la actualización y mantenimiento del *software* de forma sencilla y en un reducido espacio de tiempo. A partir del uso de *frameworks* basados en el patrón MVC se puede lograr una mejor organización del trabajo y mayor especialización de los desarrolladores y diseñadores [16].

2.6.2 Definición de las partes

El Modelo es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformación, no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. La Vista es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa preferentemente con el Controlador, pero es posible que trate directamente con el Modelo a través de una referencia al propio Modelo. Por otra parte, el Controlador es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo, centra toda la interacción entre la Vista y el Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo [16].

En el caso de la solución planteada, se utiliza el Modelo-Vista, se prescinde del Controlador, ya que, las clases de modelo / vista se pueden separar en los tres grupos descritos anteriormente: modelos, vistas y delegados. Cada uno de estos componentes se define por las clases abstractas que proporcionan interfaces comunes y en algunos casos, las implementaciones por defecto de características.

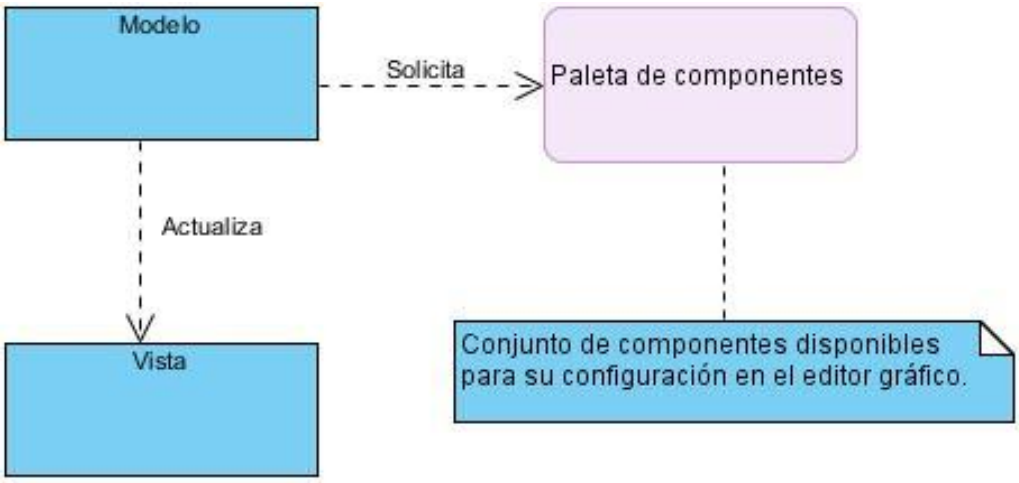


Fig. 5 Modelo-Vista-Controlador

2.6.3 Patrones de diseño

Los patrones de diseño son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas en la programación orientada a objetos. Un patrón de diseño proporciona un esquema para refinar sus subsistemas o componentes, o las relaciones entre ellos. Describe la estructura de la solución de un problema que aparece repetidamente de componentes que se comunican entre ellos [15].

Para la solución de la aplicación se utilizaron los patrones de diseño Banda de los Cuatro (*GoF*, por sus siglas en inglés), el patrón *GoF* utilizado fue el patrón de comportamiento, a continuación, se describen los mismos clasificados en tres grupos, los cuales se muestran:

- ✓ **Patrones de comportamiento:** Se utilizan a la hora de definir como las clases y objetos interaccionan entre ellos.

- ✓ **Patrones creacionales:** Empleados para instanciar objetos, y así separar la implementación del cliente de la de los objetos que se utilizan. Con ellos se intenta separar la lógica de creación de objetos y encapsularla.

- ✓ **Patrones estructurales:** Utilizados para crear clases u objetos incluidos dentro de estructuras más complejas.

Para la asignación general de responsabilidades en el *software* existen patrones denominados Patrones Generales de Asignación de Responsabilidades (GRASP, por sus siglas en inglés), que describen los principios fundamentales de asignación de responsabilidades a objetos, expresados como patrones. A continuación, se exponen algunos patrones utilizados dentro del contexto de la solución a implementar:

- ✓ **Experto**: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. De forma general en el diseño del sistema se basa en asignar a cada clase la responsabilidad que solo ellas pueden realizar. Por lo que en todas las clases del sistema utilizan este patrón.

- ✓ **Creador:** Permite asignar el responsable de la creación de una nueva instancia de alguna clase. Explica qué clase es la encargada de crear objetos, en determinados escenarios de ejecución y guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito general de este patrón es encontrar un creador que se debe conectar con el objeto producido.

- ✓ **Alta cohesión:** Facilita la solución al problema ¿cómo mantener manejable la complejidad?, mediante la asignación de responsabilidades de manera que la información almacenada en una clase sea coherente y esté relacionada con la clase. El uso de este patrón se ve reflejado en todo el diagrama debido a que cada clase almacena la información de ella misma de manera coherente.

- ✓ **Bajo acoplamiento:** ¿Cómo dar soporte a las bajas dependencias y al incremento de la reutilización? Asigne responsabilidades de manera que el acoplamiento (innecesario) se mantenga bajo.

- ✓ **Plantilla (*Template Method*):** Este patrón define una operación, la estructura interna de un algoritmo, delegando en las subclases algunos de sus atributos. Permite que las clases hijas redefinan ciertos atributos del algoritmo sin cambiar su estructura. Es utilizado en clases abstractas donde el código común será usado por las clases que heredan de ellas permitiendo la reescritura de determinados métodos. Ejemplo de dicha reestructuración y uso son los métodos *paintRuntime()*, *loadSVG()* y *paintEdition()*.

2.7 Conclusiones parciales

En el capítulo, se trataron temas referentes a la descripción de la solución de los componentes para la representación de motores eléctricos en el Editor Gráfico del *HMI* del *SCADA* SAINUX 2.0, lo que permitió arribar a las siguientes conclusiones:

- ✓ Permitió la confección del modelo del dominio, estableciendo un punto de partida para lograr un correcto diseño y entendimiento del sistema.
- ✓ Se especificó la captura de los requisitos funcionales y no funcionales, permitiendo identificar las funcionalidades con las que contará la aplicación, dándole respuesta a las necesidades del cliente.
- ✓ Se definió la arquitectura con la que contará el conjunto de componentes gráficos, empleándose el patrón arquitectónico Modelo-Vista.
- ✓ La creación del Diagrama de Clases, aportó una clara perspectiva de la implementación del sistema a llevar a cabo en la solución del mismo.
- ✓ Se identificaron los patrones de diseño a emplear en la propuesta de solución, quedando así más claro el diseño estructural del funcionamiento del sistema.

Capitulo III. Implementación y pruebas.

3.1 Introducción

Una vez diseñados los componentes gráficos y exportados al formato SVG, es necesaria su implementación para ser mostrados en la paleta de componentes del editor gráfico. Luego de culminada dicha operación, se le hacen pruebas al sistema, pues a pesar del correcto funcionamiento, puede presentar fallas en el desempeño de las funcionalidades. Este capítulo está centrado en la implementación y las pruebas al sistema, donde será desarrollada y validada la solución propuesta.

3.2 Diagrama de Componentes

Un componente es una parte física de un sistema (módulo, base de datos, programa ejecutable). Se puede decir que un componente es la materialización de una o más clases, porque una abstracción con atributos y métodos pueden ser implementados en los componentes. Los componentes pueden ser archivos, código fuente + cabeceras, librerías compartidas (*DLLs*), ejecutables o paquetes [15].

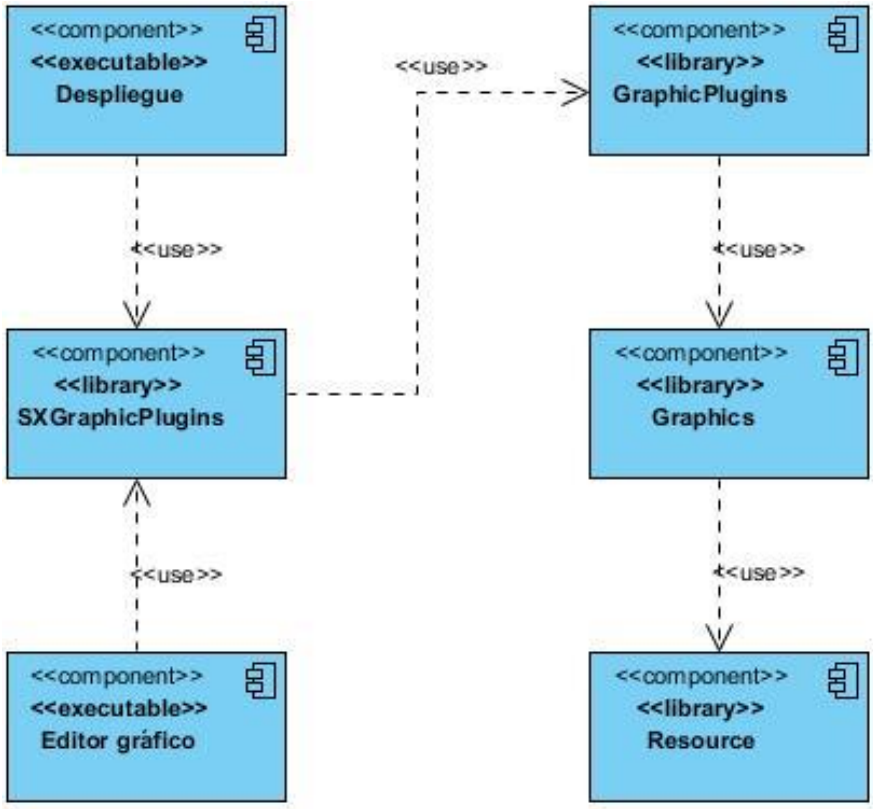


Fig. 6 Diagrama de Componentes

El diagrama de componentes anteriormente representado, está estructurado por ejecutables y bibliotecas. A continuación, se describe cada componente expuesto:

Despliegue: Aplicación donde se supervisa el área configurada para interactuar con los componentes gráficos.

GraphicPlugins: Biblioteca en la cual se tienen almacenados la lógica de los objetos gráficos.

SxGraphicsPlugin: Interfaz de la biblioteca *GraphicPlugins*, dicha biblioteca es específica para los componentes de SAINUX2.0.

Editor gráfico: Aplicación donde se trabaja en el entorno de configuración para representar los procesos del campo.

Graphics: Biblioteca donde están almacenados los objetos gráficos que se utilizan para representar los componentes gráficos, ya sea de forma simple o componentes complejos.

Resource: Biblioteca que agrupa las entidades encargadas de brindar las imágenes, los iconos y los diseños de los componentes en SVG.

3.3 Diagrama de Despliegue

El diagrama de despliegue es un diagrama que se utiliza para modelar el *hardware* utilizado en las implementaciones de sistemas y las relaciones entre sus componentes, permitiendo modelar la disposición física o topología de un sistema. Muestra el *hardware* usado y los componentes instalados en el mismo, así como las conexiones físicas entre el *hardware* y las relaciones entre componentes representados por nodos [15].

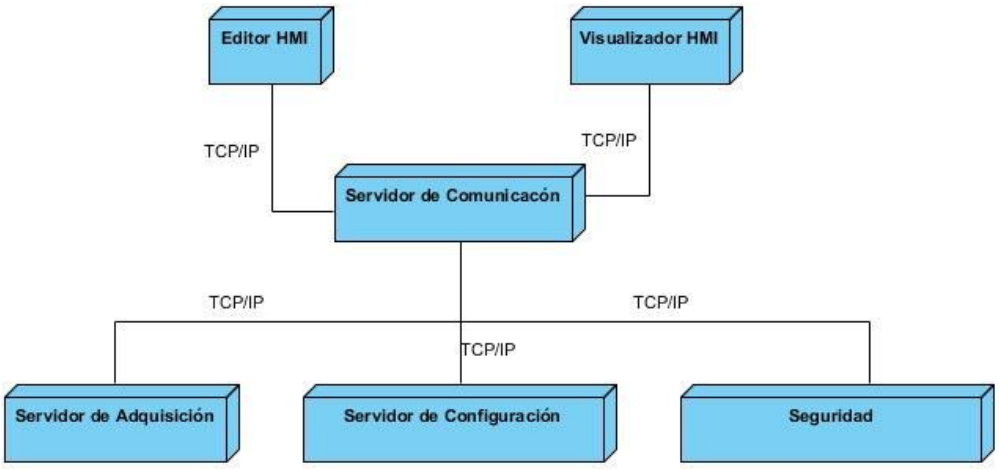


Fig. 7 Diagrama de Despliegue

3.4 Estándar de codificación

Los estándares de codificación, también conocidos como estilos de programación o convenciones de código, son convenios para escribir código fuente en ciertos lenguajes de programación. Permiten que el código en consecuencia sea sostenible y que todos los participantes lo puedan entender en un menor tiempo. Para la implementación del componente gráfico es necesario utilizar el estándar de codificación de C++ para el proyecto *SCADA* SAINUX 2.0 [17].

Algunas de las pautas que define el estándar utilizado define:

- ✓ En los archivos cabecera debe incluir el *copyright* y la licencia, o una referencia de la misma, al estilo GNU GPL.
- ✓ Se adopta el estilo de bloques de documentación de JavaDoc, el cual consiste de un bloque de comentario de estilo C.
- ✓ Para hacer una descripción breve se adopta el uso del comando @brief.
- ✓ Es importante especificar el nombre del autor y la fecha de creación de cualquier estructura en un código, para ello se utilizan los comandos @autor y @date para el nombre del autor y la fecha respectivamente.
- ✓ Para hacer referencia a otras clases utilizar el comando @see.
- ✓ El código será escrito en inglés y la documentación en español.
- ✓ Las variables de un solo carácter están permitidas sólo para contadores o temporales, cuando su propósito es obvio.
- ✓ Las variables y funciones comienzan con letra minúscula. Cada palabra consecutiva en el nombre comienza con letra mayúscula.
- ✓ Las funciones virtuales de una clase no podrán ser llamadas desde su destructor ni desde alguno de sus constructores.

3.5 Pruebas

Las pruebas son importantes en la obtención de un producto de alta calidad y buen funcionamiento, por lo que su objetivo fundamental es verificar y validar que realmente el *software* realice lo que está planificado que haga. Forman un punto indispensable para ver si una aplicación es realizada con calidad y para reducir el número de errores que no fueron detectados en la etapa de implementación. La fase de pruebas es una de las fases fundamentales del desarrollo de una aplicación. El objetivo de cada una de las pruebas es el detectar errores basándose en técnicas y estrategias empleadas en cada una de las pruebas. Dentro de las

estrategias de pruebas que existen pueden ser mencionadas las pruebas unitarias, las pruebas de integración, las pruebas de sistema, las pruebas de aceptación y pruebas de regresión.

3.5.1 Pruebas de caja negra

Las pruebas de caja negra son pruebas funcionales dedicadas a observar en el exterior de lo que se prueba. Se centran principalmente en lo que se quiere de un módulo, *charter* o sección específica de un *software*, es decir, es una manera de encontrar casos específicos en ese módulo que atiendan a su especificación. Estas pruebas se denominan de varias formas, pruebas de caja "opaca", pruebas de entrada/salida, pruebas inducidas por datos, los sinónimos son muchos y muy variados [18].

Estas pruebas permiten encontrar:

- ✓ Funciones que estén incorrectas o ausentes.
- ✓ Errores de interfaz.
- ✓ Errores en estructuras de datos o en accesos a las bases de datos externas.
- ✓ Errores de rendimiento.
- ✓ Errores de inicialización y terminación.

Las pruebas funcionales, son aquellas que se le efectúan al sistema, donde el usuario mediante las Historias de Usuario prueba cada una de las funcionalidades descritas en las mismas, el artefacto generado, son los casos de prueba. Dicha prueba es la utilizada ya que es la definida para las pruebas en el proyecto SAINUX 2.0.

3.5.3 Diseño de casos de prueba

Tabla 9 Diseño de casos de prueba

Descripción general				
Mostrar e interactuar con los componentes tipo motor eléctrico en la paleta de componentes y el despliegue.				
SC1 Mostrar los componentes en la paleta de componentes.				
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**	**Resultado de la prueba**
EC1.1 Mostrar los componentes en la paleta	1. En la paleta de componentes del editor gráfico, se visualizarán los motores eléctricos.	2. Muestran los componentes en la paleta.	Desde el editor/Pestaña Diseño/Motores eléctricos	Correcto con respecto a la respuesta del sistema

SC2 Arrastrar el componente hacia el despliegue.				
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**	**Resultado de la prueba**
EC2.1 Arrastrar componentes al despliegue	1. Desde la paleta de componentes, se selecciona el componente presionando clic sobre él y arrastrándolo hacia el despliegue liberando el clic.	2. Se arrastra correctamente el componente tipo motor eléctrico desde la paleta de componentes.	- En la paleta de componentes situada encima de la interfaz principal, se muestran en la región derecha los componentes. - En la región central de la interfaz principal, se muestra el despliegue.	Correcto con respecto a la respuesta del sistema
SC3 Mostrar componente en el despliegue				
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**	**Resultado de la prueba**
EC3.1 Mostrar componentes	1. Se mostrará el componente añadido al despliegue.	2. Se muestra el componente tipo motor eléctrico en el despliegue.	Desde el Visualizador gráfico/Sumario de despliegues/Seleccionar despliegue dando doble clic en el deseado en la lista.	Correcto con respecto a la respuesta del sistema
SC4 Eliminar el componente motor eléctrico en el despliegue.				
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**	**Resultado de la prueba**
EC4.1 Componente seleccionado en el despliegue	1. Se eliminará el componente seleccionándolo haciendo clic en él y se presiona la tecla *Supr.*	2. Se elimina el componente correctamente del despliegue.	Despliegue	Correcto con respecto a la respuesta del sistema
EC4.2 Componente seleccionado en el despliegue/Clic secundario/Eliminar	1. Se eliminará el componente seleccionándolo haciendo clic en él y se presiona clic secundario.	2. Se despliega un menú de opciones. 4. Se elimina el componente correctamente del Despliegue.	- Despliegue - Presionando clic secundario se despliega el menú de opciones del componente.	Correcto con respecto a la respuesta del sistema

	3. Se selecciona “Eliminar”.			
SC5 Modificar el tamaño del componente				
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**	**Resultado de la prueba**
EC5.1 Modificar tamaño	1. Se modificará el tamaño seleccionando el componente haciendo clic en él. 3. Se contrae o estira el componente, haciendo clic en un punto y arrastrando, luego liberar.	2. Aparecerán 6 puntos de edición en el contorno del componente. 4. Se modifica el tamaño del componente.	Siguiendo la ruta: - Despliegue - Presionando clic primario se muestran los puntos de edición de tamaño.	Correcto con respecto a la respuesta del sistema
EC5.2 Modificar tamaño/Visor de Propiedades	1. Se modificará el tamaño seleccionando el componente haciendo clic en él, luego presionar clic secundario. 3. Se da clic en la opción “Propiedades”. 5. Se le asigna manualmente los valores de ancho y alto haciendo clic en el *textbox* al lado de “ancho” y “alto”.	2. Se despliega el menú de opciones. 4. Se muestra el Inspector de propiedades. 6. Se modifica el tamaño del componente.	Siguiendo la ruta: - Despliegue - Presionando clic secundario se despliega el menú de opciones del componente. - En la región derecha de la interfaz principal, se muestra el “Inspector de propiedades”	Correcto con respecto a la respuesta del sistema
SC6 Configurar las propiedades del componente.				
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**	**Resultado de la prueba**
EC6.1 Configurar propiedades/Nombre	1. Se modificará el tamaño	2. Se despliega el menú de opciones.	Siguiendo la ruta: - Despliegue	Correcto con respecto a la

	seleccionando el componente haciendo clic en él, luego presionar clic secundario. 3. Se da clic en la opción "Propiedades". 5. Se da clic en "alias". 7. Escribir manualmente el nuevo nombre del componente.	4. Se muestra el Inspector de propiedades. 6. Se muestra el nombre del componente seleccionado para editar. 8. Se visualiza el nuevo nombre al componente	- Presionando clic secundario se despliega el menú de opciones del componente. - En la región derecha de la interfaz principal, se muestra el "Inspector de propiedades" - En la región superior del visor de propiedades se muestra "alias".	respuesta del sistema
EC6.2 Configurar propiedades/Descripción	1. Se modificará el tamaño seleccionando el componente haciendo clic en él, luego presionar clic secundario. 3. Se da clic en la opción "Propiedades". 5. Se da clic en "Descripción". 7. Escribir manualmente la descripción del componente.	2. Se despliega el menú de opciones. 4. Se muestra el Inspector de propiedades. 6. Se muestra la descripción vacía del componente seleccionado para editar. 8. Se visualiza la descripción del componente	Siguiendo la ruta: - Despliegue - Presionando clic secundario se despliega el menú de opciones del componente. - En la región derecha de la interfaz principal, se muestra el "Inspector de propiedades" - En la región superior del visor de propiedades, segunda columna se muestra "Descripción".	Correcto con respecto a la respuesta del sistema
SC7 Configurar color de las líneas de contorno del componente.				
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**	**Resultado de la prueba**
EC7.1 Configurar color de líneas	1. Se cambiará el color del componente seleccionado	2. Se despliega el menú de opciones.	Siguiendo la ruta: - Despliegue - Presionando clic secundario se despliega	Correcto con respecto a la respuesta del sistema

	haciendo clic en él, luego presionar clic secundario. 3. Se da clic en la opción "Propiedades". 4. Se da clic en "color de líneas". 6. Se selecciona el color deseado dando clic en el mismo. 7. Se da clic en el botón "Aceptar".	4. Se muestra el Inspector de propiedades. 5. Aparece paleta de colores disponibles. 8. Se muestra en el componente el nuevo color de líneas asignado.	el menú de opciones del componente. - En la región derecha de la interfaz principal, se muestra el "Inspector de propiedades" - En la región central de la interfaz principal, se muestra la "Paleta de colores"	
EC7.1 Configurar color de líneas/ Inspector de propiedades	1. Se cambiará el color del componente seleccionado haciendo clic en él 2. Se da clic en "color de base". 4. Se selecciona el color deseado dando clic en el mismo. 7. Se da clic en el botón "Aceptar".	3. Aparece paleta de colores disponibles. 5. Se muestra en el componente el nuevo color de líneas asignado.	Siguiendo la ruta: - Despliegue - En la región derecha de la interfaz principal, se muestra el "Inspector de propiedades" - En la región central de la interfaz principal, se muestra la "Paleta de colores"	Correcto con respecto a la respuesta del sistema

Los resultados obtenidos después de aplicar las pruebas funcionales al sistema son los siguientes:

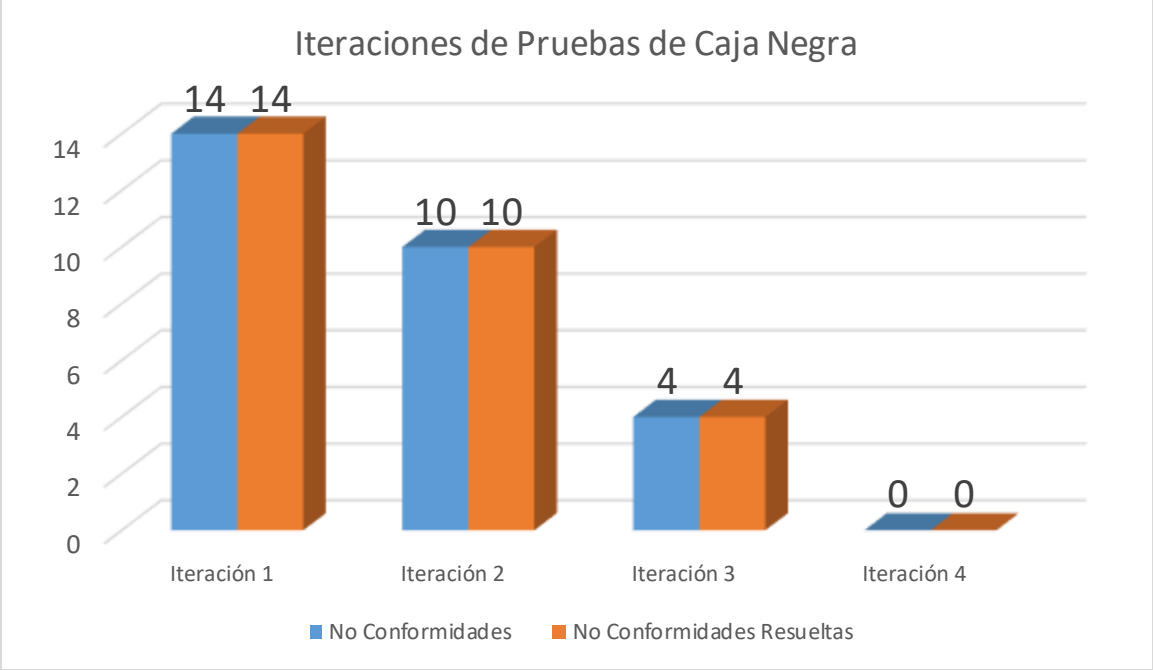


Fig. 8 Iteraciones de Pruebas de Caja negra

En la Figura 17, se muestran las iteraciones por las cuales pasó el sistema al aplicar las pruebas funcionales al sistema, las mismas dejaron evidencia de la realización de 4 iteraciones de prueba donde se detectaron 14 No Conformidades (NC) durante la primera iteración, 10 en la segunda, 4 en la tercera y en la cuarta no fue detectada ninguna NC. Dentro de los errores más comunes detectados fue el referente al requisito funcional “RF3: Mostrar componente en el despliegue", los cuales fueron corregidos, evidenciado en la fase de prueba.

El efecto de las pruebas realizadas al *software* fue satisfactorio. Se comprobó que las respuestas del sistema fueran las esperadas y que el mismo cumple con las especificaciones definidas a partir de los requisitos funcionales planteados con anterioridad. Con las pruebas aplicadas, se comprobó que el sistema cumple con los objetivos propuestos con el desarrollo de los componentes gráficos de tipo motores eléctricos en el editor gráfico del *SCADA* SAINUX 2.0.

3.6 Conclusiones parciales

En el capítulo, se trataron temas referentes a la implementación desarrollada y las pruebas realizadas a los componentes de tipo motor eléctrico en el editor gráfico del *SCADA* SAINUX 2.0, lo que facilitó arribar a las siguientes conclusiones:

- ✓ Proveyó la exposición de los principales artefactos del modelo de implementación, mostrando así los componentes del sistema y su relación a través del diagrama de componentes.
- ✓ Se modeló la arquitectura en tiempo de ejecución mediante los vínculos de comunicación que existe entre los nodos del diagrama de despliegue.
- ✓ Se finalizó con un enfoque global de cómo quedó implementado el sistema, demostrando que la aplicación cumple con las funcionalidades previstas. (Ver anexos)
- ✓ La realización de las pruebas de caja negra determinó que el sistema cumple con el objetivo general de la investigación.
- ✓ Las iteraciones de pruebas facilitaron la identificación de las no conformidades, así como la estructura para resolver las mismas sin que quedara pendiente algún mal funcionamiento.

Conclusiones generales

Con la realización de este trabajo se desarrollaron componentes gráficos para la representación de componentes gráficos de tipo motores eléctricos para la Interfaz Hombre-Máquina del *SCADA* SAINUX 2.0. De esta forma se le da cumplimiento al objetivo propuesto al inicio de la investigación, además se comprobó que:

- ✓ La metodología de desarrollo de *software* utilizada, permitió la correcta definición de las Historias de Usuario, logrando así ver con claridad los requisitos funcionales.
- ✓ Con la implementación de la solución, Componentes gráficos para la representación de motores eléctricos en el *SCADA* SAINUX 2.0, se eliminan los problemas descritos en la situación problemática.
- ✓ Mediante la realización de las pruebas funcionales al sistema, se comprobó que la propuesta de solución cumple con los requerimientos pactados con el Centro.

Recomendaciones

Teniendo en cuenta los resultados obtenidos en esta investigación se recomienda:

- ✓ Ampliar la gama de componentes gráficos de representación eléctrica.
- ✓ Aumentar los componentes gráficos en la paleta de componentes de tipo motores eléctricos, tales serían los sincromotores y motores eléctricos trifásicos.

Referencias bibliográficas

1. Boyer, S.A., *SCADA supervisory control and data acquisition*. 2018: The Instrumentation, Systems and Automation Society.
2. Iec, *IEC - Members & Experts > Reference documents: Statutes / Directives / Agreement %U http://www.iec.ch/members_experts/refdocs/governing.htm*. 2018.
3. Pérez-López, E., *Los sistemas SCADA en la automatización industrial.* Revista Tecnología en Marcha, 2015. 28(4): p. 3–14 %U http://revistas.tec.ac.cr/index.php/tec_marcha/article/view/2438.
4. Dulău, L.I., M. Abrudean, and D. Bică, *SCADA simulation of a distributed generation system with storage technologies.* Procedia Technology, 2015. 19: p. 665–672 %U https://www.sciencedirect.com/science/article/pii/S221201731500095X.
5. Pillajo, C. and J.E. Sierra, *Human Machine Interface HMI using Kinect sensor to control a SCARA Robot*, in *Communications and Computing (COLCOM), 2013 IEEE Colombian Conference on*. 2013, IEEE. p. 1–5 %U http://ieeexplore.ieee.org/abstract/document/6564822/.
6. Lobosco, O.S., J.L.P. da Costa Dias, and D. Oliver, *Selección y aplicación de motores eléctricos*. 1990: Marcombo %U http://redbiblio.unne.edu.ar/pdf/0601-006077_I.pdf.
7. Team, I., *Inkscape: A vector drawing tool.* URL http://www. inkscape. org, 2004.
8. Hunter, J.D., *Matplotlib: A 2D graphics environment.* Computing in science & engineering, 2007. 9(3): p. 90–95 %U http://aip.scitation.org/doi/pdf/10.1109/MCSE.2007.55.
9. Sousa, F.R.M., L.C. Cordeiro, and E.B. de Lima Filho, *Bounded model checking of C++ programs based on the Qt Framework*, in *Consumer Electronics (GCCE), 2015 IEEE 4th Global Conference on*. 2015, IEEE. p. 179–180 %U http://ieeexplore.ieee.org/abstract/document/7398699/.
10. Rischpater, R., *Application Development with Qt Creator*. 2014: Packt Publishing Ltd %U https://www.google.com/books?hl=es&lr=&id=NMeiBQAAQBAJ&oi=fnd&pg=PT14&dq=qt+creator&ots=EIvQvKMCMn&sig=rdiuR8QjpP3le-weSzBS6Y_u2pk.
11. Sparks, G., *Introducción al modelado de sistemas de software usando el Lenguaje Unificado de Modelado (UML) el modelo de proceso de negocio.* Enterprise Architect, Solus-Craftware Consultores Ltda, 2013: p. 1–10.
12. Inc. Paradigm, V., *Visual paradigm for uml.* Visual Paradigm for UML-UML tool for software application development, 2013: p. 72.
13. Tamara Rodríguez, S., *Metodología de Desarrollo para la Actividad Productiva de la UCI %U https://excriba.prod.uci.cu/page/context/shared/document-details?nodeRef=workspace://SpacesStore/a622adab-eac5-4fb3-ba08-a266767fff5f*. 2015.
14. Sommerville, I., *Ingeniería de software novena edición*. 2011: México: Pearson.

15. Pressman, R.S., *Software engineering: a practitioner's approach*. 2005: Palgrave Macmillan %U https://books.google.es/books?hl=es&lr=&id=bL7QZHtWvaUC&oi=fnd&pg=PR27&dq=pressman+&ots=O7vedVwJbi&sig=H68MdJfo73zNyxM-CgEPYGlhe4M.
16. Yanette Díaz, G. and R. Yenisleidy Fernández, *Patrón Modelo-Vista-Controlador.* 2012. 11(1).
17. Ariel Chávez, L. and E.G.H. Luis, *Estándares de codificación para C++ Proyecto SCADA Guardián del ALBA*. 2010.
18. Mera Paz, J.A. and others, *Análisis del proceso de pruebas de calidad de software %U http://repository.ucc.edu.co/handle/ucc/962.* 2016.
19. Cadavid, A.N., J.D.F. Martínez, and J.M. Vélez, Revisión de metodologías ágiles para el desarrollo de software. Prospectiva, 2013. 11(2): p. 30–39 %U http://www.redalyc.org/pdf/4962/496250736004.pdf.
20. Escobar-Sánchez, M.E. and W.M. Fuertes-Díaz, Modelo formal de pruebas funcionales de software para alcanzar el Nivel de Madurez Integrado 2. Revista Facultad de Ingenieria, 2015. 24(39): p. 31 %U http://search.proquest.com/openview/6ed4b266c283205c8b2e023f35f78947/1?pq-origsite=gscholar&cbl=2043296.
21. Jústiz-Núñez, D., D. Gómez-Suárez, and M.D. Delgado-Dapena, Proceso de pruebas para productos de software en un laboratorio de calidad. Ingeniería Industrial, 2014. 35(2): p. 131–145 %U http://scielo.sld.cu/scielo.php?pid=S1815-59362014000200003&script=sci_arttext&tlng=pt.
22. Raydel Raúl Viñolo, S. and F. Alexander Roquero, Sistema Gestor de Procesos de Media v2. 2012. 5(7 %U https://publicaciones.uci.cu/?journal=SC&page=article&op=view&path%5B%5D=967).
23. Miño Montaño, F.P., Análisis de motores en vehículos eléctricos %U http://repositorio.uisek.edu.ec/handle/123456789/2164. 2016.
24. Recanzone, R.R., et al., Experiencias en la enseñanza de la Informática Industrial en una carrera de Ingeniería Electrónica %U http://sedici.unlp.edu.ar/handle/10915/27528, in VIII Congreso de Tecnología en Educación y Educación en Tecnología. 2013.
25. Camargo, C., L.K. Duran, and N.F. Rosas, Plataforma hardware/software abierta para aplicaciones en procesos de automatización industrial. Ingenium Revista de la facultad de ingeniería, 2013. 14(28): p. 76–85 %U http://revistas.usb.edu.co/index.php/Ingenium/article/view/1335.
26. Chacón Morales, R.S., Simulación SCADA (Control, supervisión y adquisición de datos) de una planta generadora de energía eléctrica a base de energía geotérmica. 2012, Universidad de El Salvador %U http://ri.ues.edu.sv/1778/1/Simulacion_SCADA_en_geotermia.pdf.
27. Cruz-Ivankovich, I., Automatización en el sistema eléctrico en Siesa para el ahorro energético mediante un sistema SCADA %U https://repositoriotec.tec.ac.cr/handle/2238/6582. 2016.

28. Dis, I.S.O., 9241-210: 2010. Ergonomics of human system interaction-Part 210: Human-centred design for interactive systems. International Standardization Organization (ISO). Switzerland, 2009.
29. Mentler, T. and M. Herczeg, Applying ISO 9241-110 dialogue principles to tablet applications in emergency medical services, in ISCRAM. 2013, Citeseer %U http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.634.5462&rep=rep1&type=pdf.
30. Collins, M.J., Scalable Vector Graphics, in Pro HTML5 with Visual Studio 2015. 2015, Springer. p. 209–235 %U https://link.springer.com/chapter/10.1007/978-1-4842-1147-2_9.