Facultad 4

Desarrollo de la animación de movimiento compuesto en el SCADA SAINUX

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

Autor: Keiger Rivera Acosta.

Tutor: Rafael Alejandro Pérez Ordoñez.

Co-Tutor: Rosmery Pedraza Ceballo.

Ciudad de La Habana, 2017.

"Año 59 de la Revolución"

"El conocimiento y la tecnología es de especial relieve en nuestra agenda, porque en él abordamos los problemas que deciden, en buena medida, el futuro de nuestros países.

Fidel Castro Ruz

Declaración de Autoría

Declaro ser el autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los _____ días del mes de _____ del año _____.

Keiger Rivera Acosta
Autor

Ing. Rafael Alejandro Pérez Ordoñez
Tutor

Ing. Rosmery Pedraza Ceballo
Co-Tutor

Síntesis de los tutores

- ✓ **Tutor:** Ing. Rafael Alejandro Pérez Ordoñez.

  Síntesis: Graduado de la Universidad de las Ciencias Informáticas, tres años de experiencia en el desarrollo de sistemas SCADA.

  Correo electrónico: *rafaelalejandro@uci.cu*

- ✓ **Co-Tutor**: Ing. Rosmery Pedraza Ceballo.

  Síntesis: Graduada de la Universidad de las Ciencias Informáticas, con un año de experiencia en ingeniería de software.

  Correo electrónico: *rceballo@uci.cu*

Dedicatoria

Quiero dedicar este trabajo a mi abuela, a mi madre y a mi padrastro que han sido las personas más importantes de mi vida, que jamás me ha dado la espalda, siempre han confiado en mí, en lo que decido, gracias a ustedes por sus consejos, confianza, por darlo todo por mi sin esperar nada a cambio pienso que sea muy importante para ustedes que pudieran verme hecho un ingeniero, le doy las gracias a dios por esta bendición y por ayudarnos a que todo esté bien, las amo.

Agradecimientos

Quisiera agradecer a todas las personas que de una forma u otra han contribuido a mi formación personal y profesional, en muchos casos, las palabras no son suficientes para expresar las dimensiones de mi agradecimiento.

A mi familia

Por estar siempre a mi lado y brindarme el apoyo necesario en cada momento. Soy dichoso de tenerlos a todos ustedes. Para ustedes todo mi agradecimiento por tanto amor y dedicación. Gracias

A mis tutores

Por haber confiado en mí para la realización de este trabajo y guiarme en el desarrollo del mismo. Gracias

A mis compañeros

Ustedes han sido durante estos cinco años mi familia más cercana en esta maravillosa escuela, gran parte de lo que soy se lo debo a ustedes. Gracias

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. Una de las principales prestaciones dentro del módulo HMI del SCADA SAINUX es el entorno de configuración, este cuenta con una biblioteca de animaciones las cuales se asocian a los componentes gráficos lográndose una representación gráfica de lo que sucede en el proceso a supervisar.

El siguiente trabajo tiene como objetivo el desarrollo de la animación de movimiento compuesto que permita variar en los ejes de coordenadas (x, y) la posición de los componentes gráficos de la Interfaz Hombre Máquina del sistema SCADA SAINUX. Como resultado se obtuvo una animación de movimiento compuesto que permite variar en los ejes de coordenadas (x, y) la posición de los componentes gráficos de la Interfaz Hombre Máquina del sistema SCADA SAINUX.

Palabras claves: Animación, HMI, SCADA.

Índice de Contenido

Índice de Tablas

Índice de figuras

Introducción

La humanidad ha transitado por varias etapas en el desarrollo tecnológico, lo que ha exigido una constante investigación científica que impulse la evolución tecnológica. La automatización de procesos industriales se ha convertido en un gran avance para el sector empresarial, con su uso se han revolucionado las producciones a gran escala, incrementándose los niveles de producción y la calidad del producto final.

Actualmente diversas industrias emplean para la automatización de sus procesos productivos los sistemas de Supervisión, Control y Adquisición de Datos (SCADA, por sus siglas en inglés). Los sistemas SCADA son utilizados para supervisar, controlar y adquirir datos de procesos industriales. Están especialmente diseñados para el control de producción, proporcionando la comunicación con los dispositivos de campo y controlando el proceso desde la pantalla del ordenador. Estos sistemas pueden estar compuestos por módulos como: manejadores, núcleo de procesamiento de datos, base de datos históricos y la Interfaz Hombre Maquina (HMI, por sus siglas en inglés).

El Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas desarrolla proyectos de este tipo, entre los cuales se encuentra el SCADA SAINUX. El SAINUX es un sistema distribuido, desarrollado sobre el sistema operativo GNU/LINUX. Está compuesto por un conjunto de módulos que intercambian información a través de la red para llevar a cabo la realización de sus tareas.

Uno de ellos, es el módulo HMI, el cual es el encargado de representar de forma gráfica en la pantalla de un ordenador los procesos bajo supervisión (1). Este módulo cuenta con un conjunto de objetos gráficos, animaciones y controles que facilitan la representación del proceso supervisado. El HMI consta de dos ambientes: Ambiente de Configuración y Ambiente de Visualización. El Ambiente de Configuración permite editar el entorno de trabajo, configurar la seguridad del sistema y definir los parámetros de las variables a supervisar, mientras que el Ambiente de Visualización es el encargado de visualizar la configuración antes realizada en el ambiente de configuración y permitirle al operador interactuar con el sistema, brindado una mejor calidad en los procesos.

Uno de los elementos más importante dentro del Ambiente de Configuración son sus componentes gráficos, ya que son los que permiten la visualización del estado del proceso en el despliegue, a estos componentes gráficos se le asocian animaciones.

Situación problemática:

El HMI posee una biblioteca de animaciones las cuales se asocian a los componentes gráficos lográndose una representación gráfica de lo que sucede en el proceso a supervisar. Dentro de la biblioteca de animaciones se encuentran la rotación, traslación, escalar, parpadeo, fluido y movimiento. La biblioteca de animaciones presenta carencia en cuanto a la animación Mover que se les asocia a los componentes gráficos. La misma solo puede mover el componente paralelo a los ejes de coordenadas (abscisas u ordenadas) y en uno de los dos sentidos con que cuenta el eje seleccionado para realizar la animación, lo que tiene como consecuencia:

- ✓ La carencia de una animación de movimiento que permita variar en los ejes de coordenadas (x, y) la posición de los componentes gráficos del SCADA SAINUX.
- ✓ Existen escenarios a automatizar que quizás no puedan ser cubiertos por el sistema con la animación de movimiento existente.

Por todo lo anteriormente expresado se define como **problema de investigación:** ¿Cómo contribuir a la visualización de los cambios de posición de un componente gráfico de forma animada en el HMI del SCADA SAINUX?

En correspondencia con el problema planteado, se formula como **objeto de estudio:**

La representación de los cambios de posición de un componente gráfico en la Interfaz Hombre Máquina de los sistemas SCADA.

Siendo el **campo de acción:** La representación de los cambios de posición de un componente gráfico en la Interfaz Hombre Máquina del sistema SCADA SAINUX.

Por lo que se propone como **objetivo general:** Desarrollar una animación de movimiento compuesto que permita variar la posición de los componentes gráficos en los ejes de coordenadas (x, y).

De acuerdo al objetivo expuesto se definen las siguientes **tareas investigativas:**

1. Elaborar el marco teórico de la investigación a través del estudio del estado del arte que existe actualmente sobre el tema.
2. Analizar los conceptos básicos asociados a las animaciones de movimiento en la Interfaz Hombre Máquina de los sistemas SCADA.
3. Investigar sobre los principales elementos que componen a las animaciones de movimiento en las Interfaces Hombre Máquina de los sistemas SCADA.
4. Realizar el levantamiento de requisitos funcionales y no funcionales.
5. Seleccionar las herramientas y tecnologías para la creación de un prototipo que solucione el problema.

6. Desarrollar un componente para la configuración de la trayectoria del movimiento compuesto.
7. Realizar las pruebas necesarias para validar el cumplimiento de los requerimientos.

Luego de obtener toda la información relacionada con el tema de la investigación se plantean como **posibles resultados:**

- ✓ Animación de movimiento que permita variar en los ejes de coordenadas (x, y) la posición de los componentes gráficos de la Interfaz Hombre Máquina del SCADA SAINUX.
- ✓ Componente para la configuración de la trayectoria del movimiento.
- ✓ Una representación gráfica de los componentes para representar los procesos de campo.

Métodos teóricos

Modelación: Posibilitó la elaboración de múltiples diagramas para un mejor entendimiento y solución del problema.

Métodos empíricos:

Revisión y consulta de la documentación: Permitió el estudio de diferentes documentos y tipos de bibliografías, posteriormente se seleccionó la información necesaria para la investigación.

Observación: Método aplicado para conocer el funcionamiento existente en los despliegues del SCADA SAINUX, mediante el comportamiento de las propiedades de los objetos gráficos y sumarios, empleados para la toma de decisiones de los operadores. Permite la recogida de información de cada uno de los conceptos y variables definidas.

El presente Trabajo de Diploma está estructurado de la siguiente forma:

Capítulo 1: “Fundamentación Teórica”. Este capítulo presenta la elaboración del marco teórico de la investigación y se presentan los argumentos teóricos que responden a las técnicas a emplear y el porqué de la necesidad de tener la animación de movimiento compuesto en la biblioteca de animaciones del SCADA SAINUX.

Capítulo 2: “Análisis y Diseño de la solución”. En este capítulo se da a conocer la propuesta del sistema y los diagramas para apoyar la comprensión del funcionamiento del mismo. Tomándose como base la metodología AUP-UCI para guiar el proceso de desarrollo.

Capítulo 3: “Implementación y Pruebas”. En este capítulo se detallan algunos aspectos de la etapa de implementación de la animación de movimiento compuesto y los diagramas correspondientes al modelo de implementación, al igual que se detallan las pruebas realizadas al sistema y los resultados obtenidos de estas.

Capítulo 1. Fundamentación teórica

1.1 Introducción

El presente capítulo tiene como objetivo definir y elaborar un marco teórico donde se expongan temas fundamentales que orienten la investigación. Para ello se describe el concepto general del SCADA, las funciones que este debe cumplir y los módulos que posee el SCADA SAINUX, del mismo modo se introduce el sistema de animaciones, así como los diferentes componentes para su representación. Además, se mencionan las principales herramientas, tecnologías y la metodología de desarrollo de software que serán empleadas para llevar a cabo el desarrollo de la solución.

1.2 Principales Conceptos

1.2.1 Automatización Industrial

Se define como automatización industrial a la aplicación de distintas tecnologías para controlar y monitorizar un proceso, máquina o dispositivo que por lo regular cumple tareas o funciones repetitivas, haciendo que operen automáticamente, reduciendo al mínimo la intervención humana. Lo que se busca con la automatización industrial es generar la mayor cantidad de producto, en el menor tiempo posible, con el fin de reducir los costos y garantizar una uniformidad en la calidad (2).

1.2.2 Sistemas SCADA

Un elemento fundamental dentro de la automatización industrial lo constituyen los sistemas SCADA. Los SCADA, son aplicaciones de software de control de producción, que se comunican con los dispositivos de campo y controlan el proceso desde la pantalla de un ordenador. Los sistemas SCADA son partes integrales de la mayoría de los ambientes industriales complejos o muy geográficamente dispersos porque pueden recoger información de grandes cantidades de fuentes y la presentan de forma más amena a los operadores. (3)

Los sistemas SCADA proveen de la información que se genera en un proceso productivo, convirtiéndose en un instrumento esencial para la toma de decisiones.

Entre las funciones básicas de un sistema SCADA se encuentran las siguientes: supervisión y control remoto de instalaciones, procesamiento de información, presentación de gráficos dinámicos, generación de reportes, presentación de alarmas, almacenamiento de

información histórica, presentación de gráficos de tendencias y programación de eventos (4).

1.2.2.1 Funciones Específicas

Transmisión: Ofrece la transferencia de datos con dispositivos de campo y otros ordenadores.

Presentación: Representación gráfica de los datos. Interfaz del Operador o HMI.

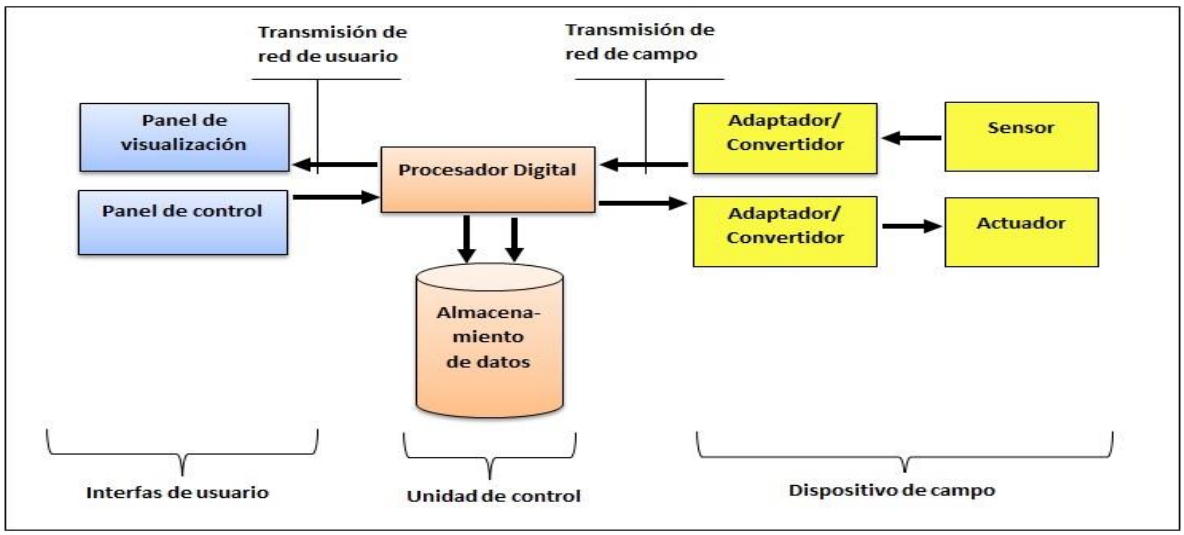


Fig.1 Esquema básico de un sistema SCADA.

1.2.3 SCADA SAINUX

SAINUX es un SCADA para el despliegue en la industria nacional, basado en tecnologías libres para su desarrollo. Este software es desarrollado por el CEDIN e 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 son los siguientes: comunicación, configuración, seguridad, base de datos históricos, adquisición y HMI, estas son sus características principales:

- ✓ **Comunicación:** Este módulo representa la capa de software encargada de la comunicación entre los diferentes procesos distribuidos, de mediano y alto nivel.
- ✓ **Configuración:** Módulo encargado de almacenar, persistir y suministrar, la información base para el funcionamiento de los demás módulos.

- ✓ **Seguridad:** Permite a los usuarios autenticarse en el sistema, y de esta forma poder acceder sólo a los recursos que tiene asignado su rol[1].
- ✓ **Bases de Datos Históricos:** Es el que permite almacenar la sucesión de alarmas y eventos, así como la información recibida desde los dispositivos que se encuentran en el campo**.** Esta información puede ser consultada y presentada mediante gráficos de tendencia y reportes.
- ✓ **Adquisición o Procesamiento de Datos:** Es el responsable de recolectar y procesar datos desde los dispositivos de campo.
- ✓ **HMI:** Representa gráficamente los procesos con los que interactúa el usuario. A través de este módulo se puede visualizar el estado de las comunicaciones con los dispositivos de campo, los valores de las variables procesadas y las alarmas generadas.

1.2.4 HMI

Muestra los componentes gráficos que facilitan la representación del proceso bajo supervisión como tuberías, válvulas, motores, permitiendo que el operador posea el control sobre los procesos que intervienen en la industria. Este módulo logra que el operario este en contacto directo con el sistema para así realizar la supervisión del proceso en general.

1.2.4.1 Ambiente de configuración del HMI.

Permite definir el ambiente de trabajo del SCADA, adaptando mejor la aplicación al proceso que se desea supervisar. Es el ambiente donde el mantenedor[2] configura la información específica del área que se desea supervisar, diseña los despliegues, los cuales haciendo uso de los componentes gráficos y las animaciones permiten simular los procesos de campo.

1.2.4.2 Ambiente de visualización del HMI.

El ambiente de ejecución se encarga de visualizar los componentes gráficos y las animaciones definidos en el ambiente de configuración. El ambiente de visualización es donde el operador puede supervisar y controlar la configuración realizada en el ambiente de configuración, comunicándose con los componentes gráficos para emitir control sobre el sistema y permitir monitorear los cambios de estado de las variables, la gestión de las alarmas y los reportes.

[1] Rol: Responsabilidad asumida por una persona en el equipo de trabajo.
[2] Mantenedor: Persona que se ocupa del mantenimiento de aparatos o servicios.

1.2.4.3 Componentes Gráficos

Los componentes gráficos se encargan de mostrar de forma más amena los procesos que se desean supervisar. Estos componentes se agrupan en:

- ✓ **Básicos:** Formado por figuras simples como líneas, flechas, elipses, rectángulos y polígonos.
- ✓ **Controles:** Compuesto por componentes que permiten ejercer control sobre el sistema tales como botón pulsable, botón de estado y lista desplegable.
- ✓ **Medidores:** Conformado por elementos que posibilitan la visualización del estado de los valores que obtienen las variables durante el proceso, entre los que se encuentran el visualizador y el medidor vertical.
- ✓ **Gráficos:** Son aquellos componentes que permiten representar el estado de los valores de una variable para realizar un análisis estadístico, entre ellos se encuentran el gráfico XY y el gráfico de barra.
- ✓ **Miscelánea:** Reúne elementos como la imagen, el diodo emisor de luz y el ventilador/extractor.
- ✓ **Tuberías:** Compuesto por aquellos elementos que forman una red de tubería, encargados del transporte de líquidos y gases.
- ✓ **Válvulas**: Reúne aquellos elementos que tienen como objetivo controlar el paso del fluido por una red de tuberías.
- ✓ **Bomba:** Formado por componentes que son utilizados para transportar líquidos y gases.
- ✓ **Almacenamiento:** Son los elementos encargados de almacenar el producto.
- ✓ **Azúcar:** Conformado por los elementos que se encargan de la administración y control de los procesos que intervienen en la industria azucarera.

Los componentes gráficos deben desarrollarse de tal manera que se le puedan aplicar animaciones y que estas sean configurables. Las animaciones son una forma de lograr dinamismo en los despliegues y representar lo que sucede en el proceso supervisado.

1.2.4.4 Animación

Una animación es el cambio visual que se produce a lo largo del tiempo. Se pueden modificar varios aspectos de un elemento gráfico para animarlo: posición, tamaño, rotación, color, y transparencia. Al ir generando cambios en la imagen, se produce en el usuario la sensación de movimiento. La animación de gráficos bidimensionales en general se basa en el concepto de fotograma o *frame* (5). De esta forma se puede decir que una animación está compuesta por una secuencia de fotogramas que son mostrados al usuario en orden y uno detrás del otro. Por lo tanto, un fotograma es un estado de los elementos que componen la animación en un instante concreto de tiempo (5). La sucesión de estos fotogramas produce la sensación de movimiento.

1.2.4.5 Animaciones en el HMI

El movimiento de figuras gráficas es muy útil cuando se crean efectos especiales con pantallas animadas en sistemas SCADA (6). Los dispositivos deben desarrollarse de tal manera que se le puedan aplicar animaciones y que estas sean configurables. Una forma de lograr dinamismo en los despliegues y representar lo que sucede en el proceso, es variando las propiedades de los objetos gráficos en dependencia del valor de una variable. Entre las animaciones que contiene el HMI se encuentran:

Visibilidad

El componente se pondrá visible cuando la variable asociada cumpla con la condición seleccionada, a tal efecto se podrá:

- ✓ Habilitar la animación.
- ✓ Seleccionar la variable o expresión que va a definir la visibilidad.
- ✓ Seleccionar la condición de comparación

Rotación

Permite cambiar el ángulo de rotación del objeto en función de una variable. Se deben definir las siguientes funcionalidades:

- ✓ Habilitar la rotación.
- ✓ Definir variable o expresión.
- ✓ Seleccionar la variable o expresión que va a definir la rotación.

Transparencia

Mediante la animación transparencia se permite modificar la transparencia de un objeto en dependencia del valor de una variable. Para lograr esta animación se deben definir una serie de parámetros como:

- ✓ Habilitación de la animación, esto debe posibilitar al usuario seleccionar si desea habilitar la animación.
- ✓ Definir la variable cuyo valor definirá transparencia del objeto.
- ✓ Posibilitar relacionar valores de la variable con determinados niveles de transparencia.

Texto asociado al componente

Por medio de esta animación es posible mostrar, en diversos formatos, la información de los valores de las variables. Se debe proporcionar la siguiente acción:

- ✓ Habilitar animación del texto.

1.2.4.6 Animación de movimiento

Mover en el eje X

Se basa en desplazar al componente gráfico en el eje horizontal del despliegue, en función del valor de una variable asociada al movimiento. Cuando la variable tome valores dentro de los puntos para los cuales fue definido el movimiento el objeto se irá moviendo desde el punto inicial hasta el final, las siguientes funcionalidades deben ser definidas:

- ✓ Habilitar animación.
- ✓ Definir si el movimiento va ser inverso o directo.
- ✓ Seleccionar la variable que va a definir la animación.
- ✓ Indicar el valor de la variable para el cual el movimiento iniciará.
- ✓ Indicar el valor de la variable para el cual el movimiento terminará.

Mover en el eje Y

Se basa en desplazar al componente gráfico, en el eje vertical del despliegue, en función del valor de una variable asociada al movimiento. Cuando la variable tome valores dentro de los puntos para los cuales fue definido el movimiento el objeto se irá moviendo desde el punto inicial hasta el final, las siguientes funcionalidades deben ser definidas:

- ✓ Habilitar animación.
- ✓ Definir si el movimiento va ser inverso o directo.
- ✓ Seleccionar la variable o expresión que va a definir la animación.
- ✓ Indicar el valor de la variable para el cual el movimiento iniciará.
- ✓ Indicar el valor de la variable para el cual el movimiento terminará.

1.2.4.7 Animación Mover y parámetros de configuración

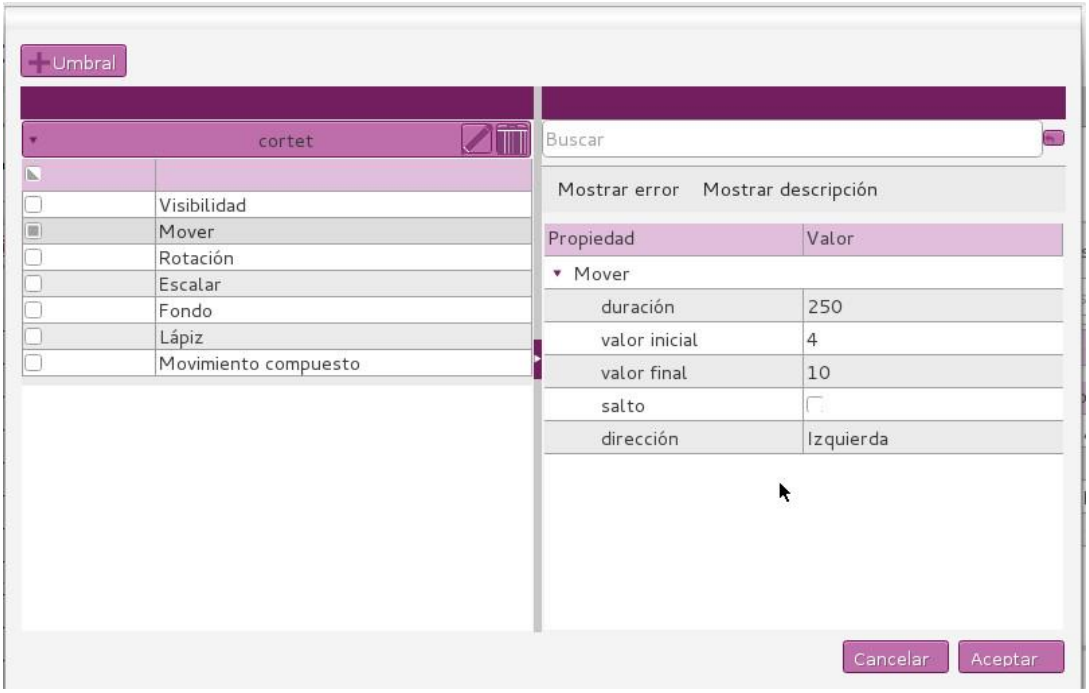


Fig.2 Animación mover y parámetros

1.2.4.8 Parámetros de Configuración

- ✓ **Duración:** Tiempo de duración especificado para la animación.
- ✓ **Valor inicial:** Valor para que el componente inicie la animación de movimiento.
- ✓ **Valor final:** Valor para que el componente finalice la animación de movimiento.
- ✓ **Salto:** Define si la animación movimiento se va a observar como un salto de un lugar al otro o si el objeto se desplazara en la dirección seleccionada durante el tiempo especificado en el parámetro duración.
- ✓ **Dirección:** Define en qué dirección se moverá el componente (izquierda, derecha, arriba o abajo).

Luego de hacer un estudio de las animaciones que posee el HMI del SCADA SAINUX se puede decir de que existen escenarios a automatizar que quizás el sistema no pueda cubrir con la animación de movimiento existente en la biblioteca, por lo que este trabajo se centra en el desarrollo de la animación de movimiento compuesto.

1.3 Soluciones de software analizadas

- ✓ **InduSoft Web Studio**

Es una herramienta que acciona en los sistemas SCADA haciendo uso de las animaciones. Permite fortalecer un amplio espectro para el control supervisado y adquisición de datos en la automatización industrial teniendo un gran dominio sobre las animaciones. La estación de trabajo va en procesión de dispositivos conectados según sus parámetros de proyecto. Las animaciones le dan a la herramienta un gran orden sobre gráficos en columna o ajuste de la escala de objetos con configuración fácil de usar. Otras animaciones que posee la herramienta incluyen "comandos" (táctil, teclado e interacción del ratón), relacionan, avisan por escrito enlace de datos, colorean, ajustan el tamaño (la anchura), cambio de posición, y rotación. (7)

- ✓ **Synfig Studio**

Synfig Studio es un software de animación 2D libre y de código abierto, basado en vectores, perfilado para la creación de animaciones**.** Las animaciones no son creadas fotograma a fotograma sino por interpolación, además de que son exportadas como GIF, aspectos que limitan la complejidad y el rendimiento de la animación final. Synfig Studio está disponible para Windows, Linux y MacOS y está desarrollado en C++. El soporte de la herramienta no es gratuito.

1.3.1 Análisis de las soluciones

Después de hacer un estudio del estado del arte de las aplicaciones para el desarrollo de las animaciones se llegó a la siguiente conclusión:

- ✓ Las soluciones analizadas son aplicaciones propietarias, con licencias costosas, a las que ni pagando su precio, Cuba puede acceder por restricciones comerciales. Además de que el Centro CEDIN sigue una línea de desarrollo de software libre por lo que soluciones privativas no tributarían a las tendencias del Centro ni de la UCI.
- ✓ Las soluciones libres estudiadas en la investigación carecen de opciones avanzadas, y solo permiten crear animaciones básicas que no satisfacen las necesidades.

1.4 Metodología y Herramientas

A continuación, se realiza una valoración de las principales herramientas que serán empleadas en la solución del problema planteado, definiendo sus características fundamentales. En algunos casos, la selección está fundamentada en su utilización por el proyecto SCADA SAINUX en la implementación de sus productos y funcionalidades.

1.4.1 Metodología de desarrollo de software

La UCI cuenta con centros productivos en su gran mayoría pertenecientes a las facultades que conforman a 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. En estas actividades productivas se utilizan una amplia variedad de metodologías, tanto ágiles como robustas y se ha comprobado que muchos proyectos no lo usan en su totalidad. (9)

1.4.2 Metodología AUP-UCI.

Para el desarrollo de la presente investigación se utiliza como metodología de desarrollo AUP-UCI, ya que está definida por el proyecto, además 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 de software de negocio usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP. (10)

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 en sus productos de software. El uso de esta técnica de modelado ágil permite encapsular los requisitos funcionales en Historias de Usuario (HU) o descripción de requisitos por procesos. (10)

1.4.3 Lenguaje de programación

Todo producto informático necesita un lenguaje de programación para su desarrollo, el cual se comunica con la computadora y orienta a la misma para que realice una tarea específica. El lenguaje de programación seleccionado es C++, con el objetivo de lograr una mayor compatibilidad con el SCADA SAINUX pues es el lenguaje en el que está desarrollado, permite el trabajo tanto a alto como a bajo nivel, logrando gran eficiencia en tiempo de ejecución y bajo consumo de memoria en los programas desarrollados.

1.4.4 Marco de trabajo

El marco de trabajo utilizado es *Qt* en su versión 4.8.2, el cual es multiplataforma para el desarrollo de aplicaciones, las cuales pueden ser con o sin interfaz gráfica (11). Para el desarrollo del presente trabajo se hace uso de este marco de trabajo debido a que utiliza el lenguaje de programación C++, es preciso para realizar todas las interfaces gráficas necesarias y está definido para utilizar en el proyecto.

1.4.5 Entorno de Desarrollo Integrado (IDE)

El IDE seleccionado para el desarrollo de la aplicación es *QtCreator* en su versión 2.5.0. *QtCreator* 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. (12) Presenta disponibilidad de código fuente, la excelente documentación organizada que provee en el *QtAssistant* y un editor para el diseño de formularios denominado *QtDesigner*.

1.4.6 Lenguaje Unificado de Modelado (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. (13)

1.4.7 Visual Paradigm

La herramienta seleccionada para realizar el modelado de la solución es Visual Paradigm en su versión 8.0. Visual Paradigm es una herramienta de Ingeniería de Software Asistida por Computación (CASE, por sus siglas en inglés). La misma propicia un conjunto de funcionalidades para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. (14)

1.4.8 Herramienta de control de versiones

Se utiliza *Apache Subversion* (SVN) en su versión 1.6. SVN es una herramienta basada en un repositorio cuyo funcionamiento se asemeja al de un sistema de ficheros. (15)

Utiliza el concepto de revisión para guardar los cambios producidos en el repositorio. Entre dos revisiones sólo guarda el conjunto de modificaciones, optimizando así al máximo el uso de espacio en disco. Permite al usuario crear, copiar y borrar carpetas con la misma facilidad con la que lo haría si estuviese en su disco duro local. Dada su flexibilidad, es necesaria la aplicación de buenas prácticas para llevar a cabo una correcta gestión de las versiones del software generado. Esta herramienta permite recuperar versiones antiguas y ver el historial de cambios de un sistema de archivos. (15)

1.5 Conclusiones parciales

- ✓ La elaboración del marco teórico brindó un acercamiento a los principales conceptos, técnicas y métodos a utilizar durante el resto de la investigación, lo que permitió profundizar los aspectos teóricos de sistemas SCADA, sus interfaces hombre-máquina y el análisis de las animaciones que el mismo posee, propiciando una base de conocimientos útil para el desarrollo de la investigación.
- ✓ La indagación sobre las técnicas y herramientas consolidaron las propuestas tecnológicas necesarias para la realización del componente, lográndose una fundamentación técnica que justifica la solución escogida.

Capítulo 2. Análisis y Diseño de la Solución

2.1 Introducción

El presente capítulo tiene como objetivo describir las actividades realizadas durante el proceso de análisis y diseño, con el objetivo modelar y dar respuesta al problema planteado. Se especifican los requisitos funcionales y no funcionales que debe cumplir el sistema para satisfacer las necesidades del cliente. Al mismo tiempo se muestra la definición de las historias de usuario que regirán el desarrollo de la solución propuesta, y se realiza la descripción del diseño.

2.2 Modelo del Dominio

Un modelo de dominio asocia todas las entidades o conceptos que se manejan en el entorno en el que trabaja el sistema mediante un diagrama de clases UML. (16)

A continuación, se presenta el modelo de dominio correspondiente a la solución, así como la descripción de los conceptos involucrados.

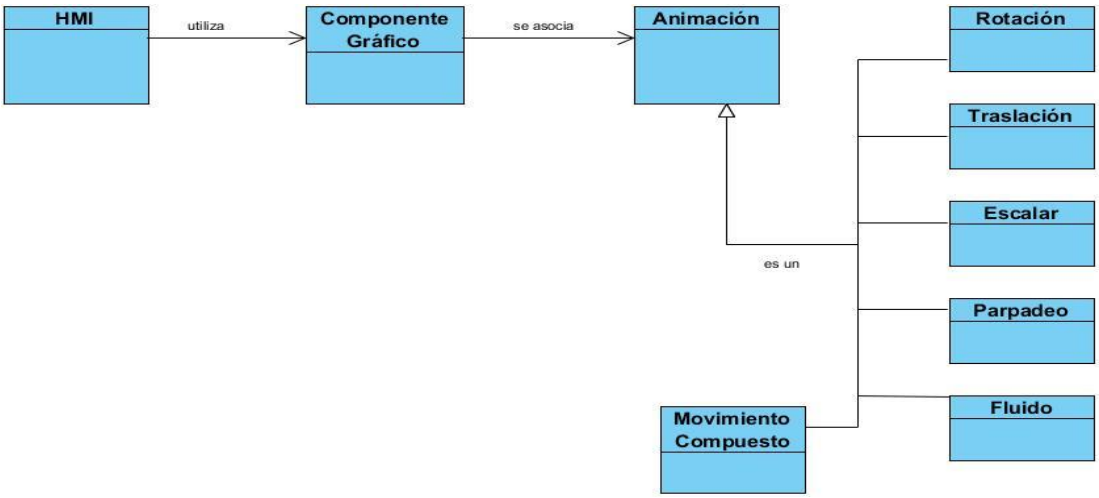


Fig. 3 Modelo de Dominio.

- ✓ **HMI:** Interfaz encargada de mostrar en un ordenador los procesos que ocurren en el campo.
- ✓ **Componente gráfico:** Representa de manera visual una serie de datos por medio de figuras o signos. Deben posibilitar la interacción de los operadores con la interfaz gráfica para de esta forma tomar decisiones respecto al proceso que se controla.
- ✓ **Animación:** Representación de movimiento de imágenes o dibujos, en este caso componentes gráficos, mediante la transformación de cada fotograma. Las

animaciones son una forma de lograr dinamismo en los despliegues y representar lo que sucede en el proceso, variando las propiedades de los objetos gráficos en dependencia del valor de una variable o de una expresión que contiene variables.

- ✓ **Movimiento Compuesto:** Animación que permite variar los componentes gráficos de posición en los ejes de coordenadas (x, y).

2.3 Propuesta de Solución

2.3.1 Animación de movimiento compuesto en el SCADA SAINUX

No solo permite posibilitar mover objetos a lo largo de trayectorias no lineales, sino que están compuestos de dos o más segmentos de línea recta.

El Movimiento Compuesto (XY) permitirá que se asocie una variable al movimiento, en función de un camino compuesto de segmentos de recta que se pueden editar libremente con el ratón, como muestra la figura.

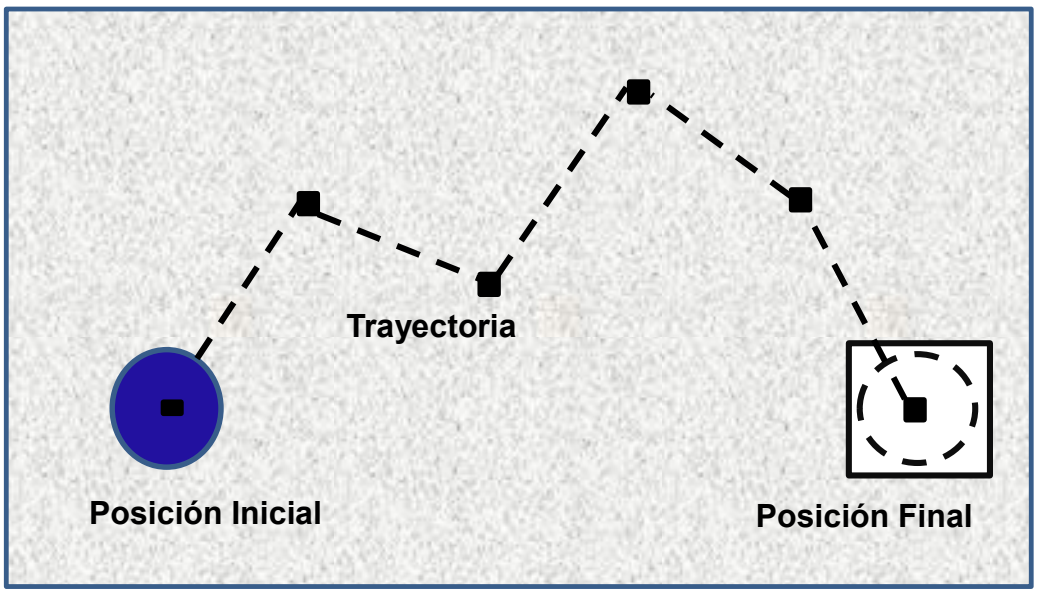


Fig. 4 Movimiento Compuesto

Para lograr esta animación se deben definir las siguientes funciones:

- ✓ Trayectoria que debe seguir el objeto.
- ✓ Habilitación de la animación, esto debe posibilitar al usuario seleccionar si desea habilitar la animación.
- ✓ Seleccionar la expresión que va a definir la animación.

2.4 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 (17). Pueden ser funcionales o no funcionales.

2.4.1 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 antes situaciones particulares (17).
A continuación, se muestran los requisitos a implementar en la propuesta de solución.

RF 1 - Asociar la animación de movimiento compuesto a los componentes gráficos.
RF 2 - Definir la trayectoria del componente gráfico.
RF 3 - Eliminar punto de la trayectoria.
RF 4 - Ejecutar la animación en el ambiente de visualización.

2.4.2 Requisitos no funcionales (RNF)

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. (17)

Requisitos del software

- ✓ El sistema debe funcionar en el Sistema Operativo Debian, en su versión 8.

Requisitos de Hardware

Debe ser ejecutado en computadoras que tengan como requerimientos mínimos los siguientes:

- ✓ Microprocesador: dual Core a 1.5 GHz.
- ✓ RAM: 2 GB

Restricciones en el diseño y la implementación

- ✓ Se implementará utilizando el lenguaje de programación C++.
- ✓ Se empleará el marco de trabajo Qt, en su versión 4.8.2.

Requisitos de usabilidad

- ✓ El sistema debe proporcionar una interfaz gráfica sencilla, intuitiva y fácil de entender.

Rendimiento

- ✓ En el ambiente de despliegue solo se admiten hasta 100 componentes.

2.4.3 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 (18). A continuación, se presentan las historias de usuarios que forman parte de la propuesta de solución.

Historia de Usuario	
Número: 1	**Nombre del requisito:** Asociar la animación de movimiento compuesto a los componentes gráficos.
Programador: Keiger Rivera Acosta.	**Iteración Asignada:** 2.
Prioridad: Alta.	**Tiempo Estimado:** 1 semana.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 1 semana.
Descripción: Posibilita al operador el cambio de algunas de las propiedades configurables del componente con el uso de animaciones. El operador debe hacer clic derecho sobre el componente, seleccionar la opción Propiedades, de esta forma se muestra el inspector de propiedades donde elige la opción animaciones y asocia la animación al componente.	

Prototipo de interfaz:

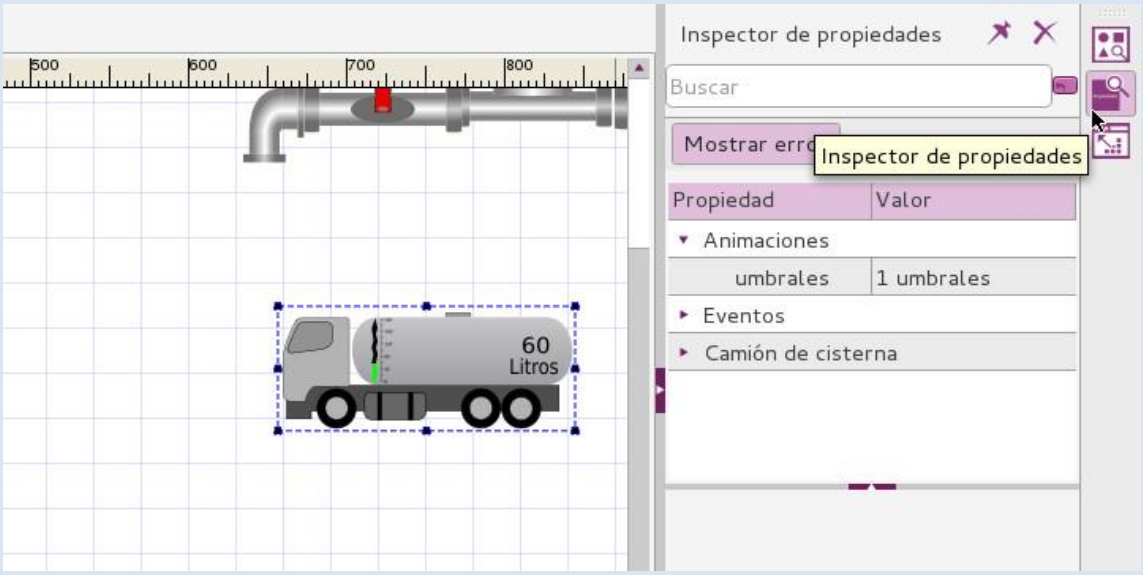


Tabla 1.HU1: Permite asociar la animación a los componentes gráficos.

Historia de Usuario	
Número: 2	**Nombre del requisito:** Definir la trayectoria del componente gráfico.
Programador: Keiger Rivera Acosta.	**Iteración Asignada:** 2.
Prioridad: Alta.	**Tiempo Estimado:** 1 semana.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 1 semana.
Descripción: Facilita al operador configurar el trayecto del componente gráfico. Parámetros de configuración para definir la trayectoria del componente gráfico. ✓ **Duración:** Define el tiempo de permanencia de los estados inicial y final durante la animación medido en milisegundos. Esta propiedad aplica únicamente si se ha seleccionado previamente la opción de parpadeo. ✓ **Modo del Movimiento:** Especifica el modo del movimiento que se quiere que realice el componente, puede ser proporcional o fraccionario.	

- ✓ **Cíclico:** Permite especificar si el movimiento que va a realizar el componente sea cíclico o no.
- ✓ **Puntos del camino:** Es parámetro que permite definir la trayectoria del componente gráfico.

El operador debe seleccionar la animación de movimiento compuesto en la colección de umbrales, ir a la opción Propiedad del movimiento compuesto, de esta forma se muestra el inspector de propiedades de la animación, seleccionar la opción puntos del camino donde configura la trayectoria del componente gráfico.

Prototipo de interfaz:

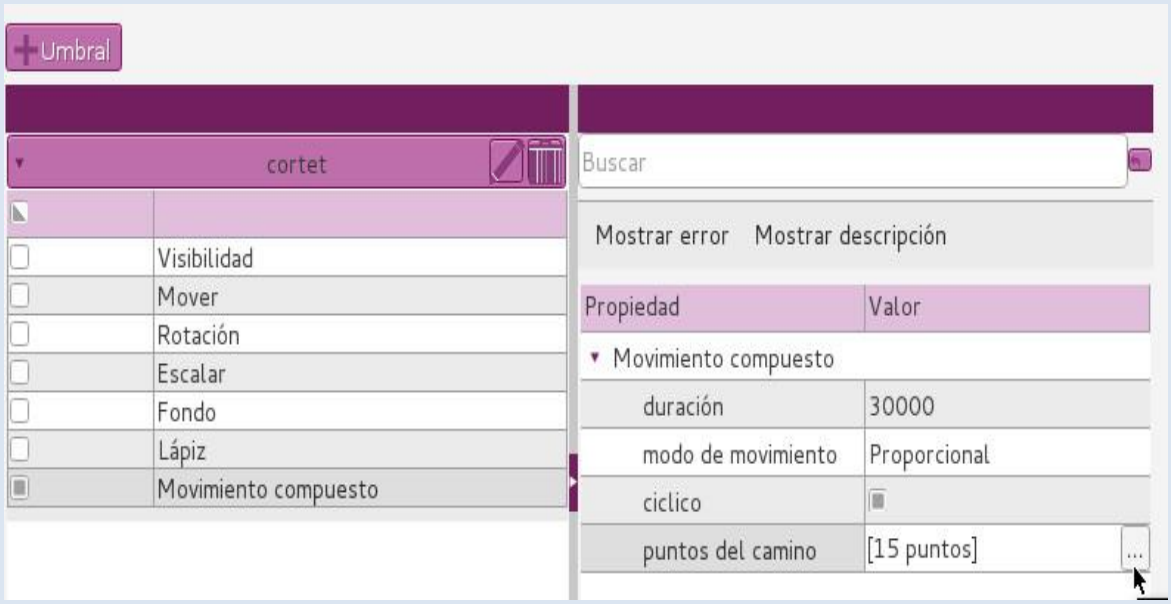


Tabla 2.HU2: Definir la trayectoria del componente gráfico.

Historia de Usuario	
Número: 3	**Nombre del requisito:** Eliminar punto de la trayectoria
Programador: Keiger Rivera Acosta.	**Iteración Asignada:** 2.
Prioridad: Alta.	**Tiempo Estimado:** 1semana.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 1 semana.

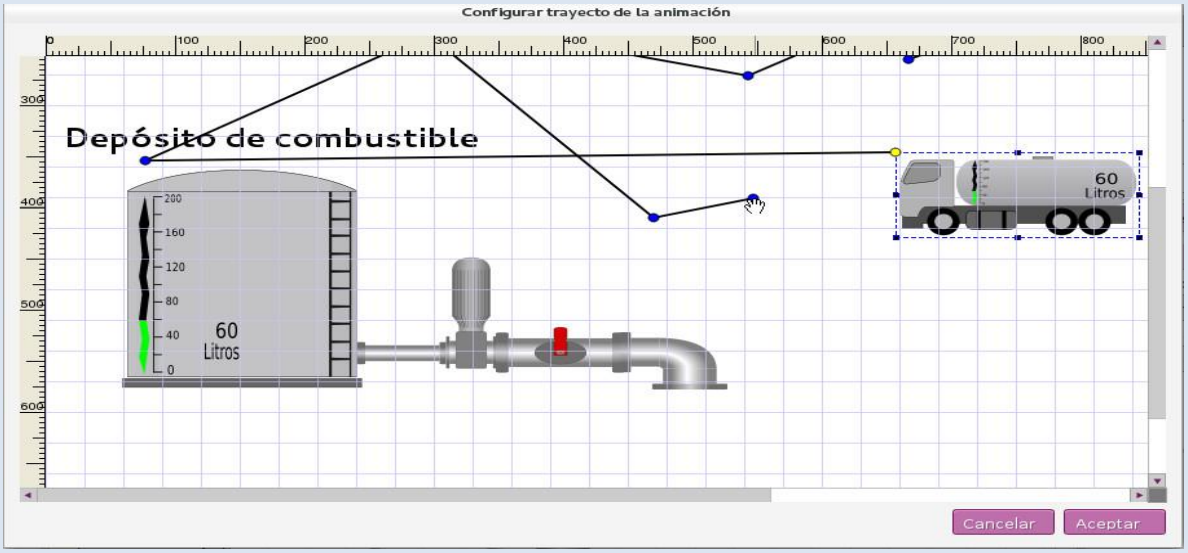
Descripción: Posibilita al operador eliminar un punto de la trayectoria del componente gráfico. El operador debe seleccionar la animación de movimiento compuesto en la colección de umbrales, ir a la opción Propiedad del movimiento compuesto, de esta forma se muestra el inspector de propiedades de la animación, seleccionar la opción puntos del camino donde configura la trayectoria del componente gráfico, clic derecho sobre el punto que desea eliminar.
Prototipo de interfaz: 

Tabla 3.HU3: Eliminar punto de la trayectoria.

Historia de Usuario	
Número: 4	**Nombre del requisito:** Ejecutar la animación en el ambiente de visualización.
Programador: Keiger Rivera Acosta.	**Iteración Asignada:** 2.
Prioridad: Alta.	**Tiempo Estimado:** 1semana.
Riesgo en Desarrollo: Riesgo definido en el GESPRO.	**Tiempo Real:** 1 semana.

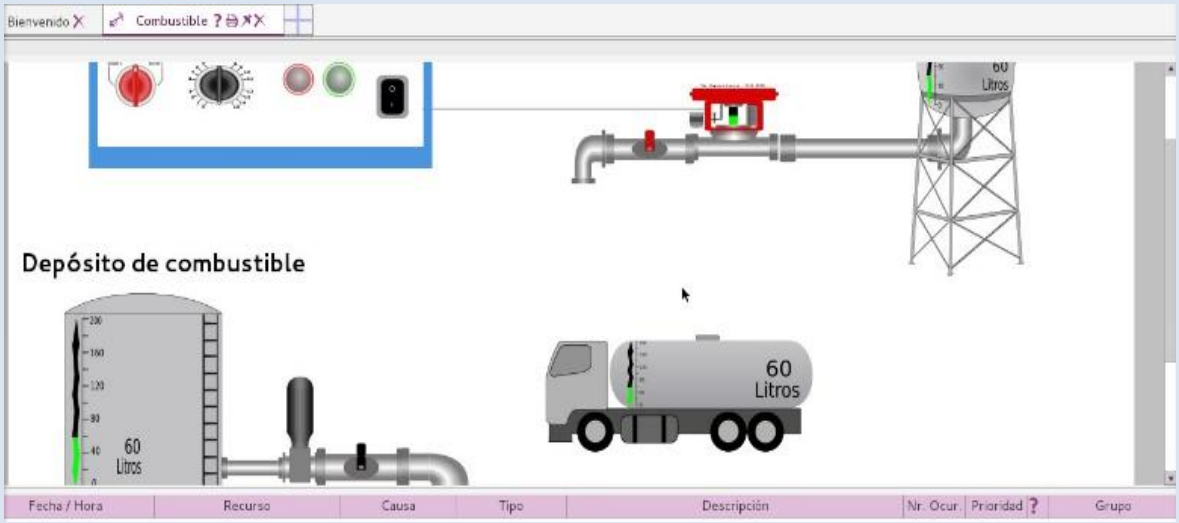
Descripción: En el despliegue se debe realizar el recorrido de la trayectoria configurada en dependencia del valor que tome la medición asociada. Cuando la medición toma valores menores que el valor inicial el componente grafico se mantiene en la posición inicial, y cuando tome valores mayores que el valor final se mantiene en la posición final.
Prototipo de interfaz 

Tabla 4.HU4: Ejecutar la animación en el ambiente de visualización.

2. 5 Descripción de la solución

La animación propuesta será incorporada al sistema de animaciones existentes actualmente al SCADA SAINUX, para de esta manera cubrir aquellos escenarios de automatización que necesiten representaciones gráficas que durante su movimiento varíen la posición de los componentes gráficos en las coordenadas (x, y) y ha de cumplir los requisitos descritos en el epígrafe anterior.

2.6 Diagrama de paquetes

Es el encargado de representar las dependencias entre los paquetes que componen un modelo. Es decir, muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre ellas. (19)

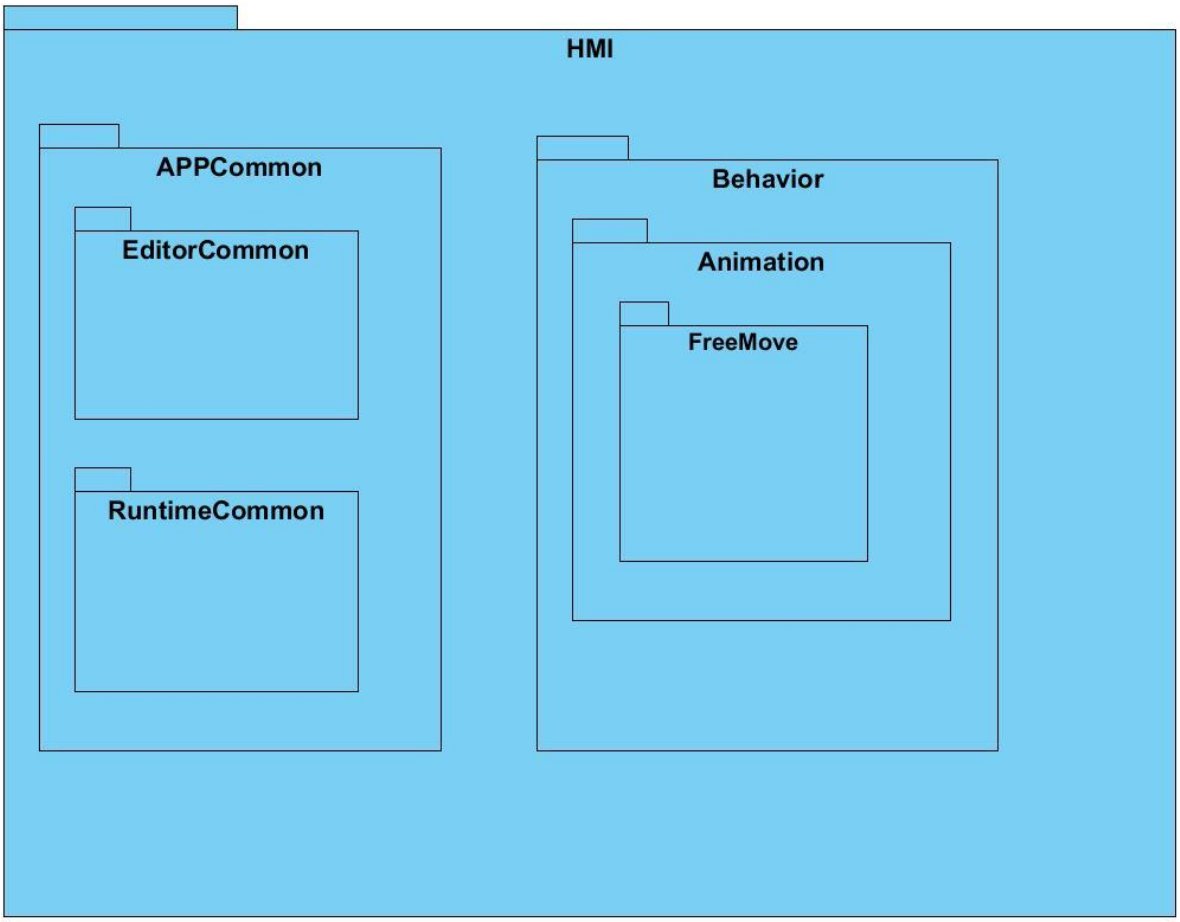


Fig. 5 Diagrama de Paquetes de la animación.

La solución propuesta esta agrupada en Componentes y Animaciones. La primera está formada por la carpeta *AppCommon*, la cual contiene a las subcarpetas *EditorCommon* y *RuntimeCommon*, que contiene las entidades referentes al Ambiente de Configuración y el Ambiente de Visualización. En la segunda parte están los paquetes *Behavior* y dentro de él el subpaquete *Animation* donde están situadas las entidades relacionadas con las animaciones que posee el sistema y como nueva animación integrada al sistema la entidad *FreeMove,* la cual va permitir que un componente realice la trayectoria definida para la animación de movimiento compuesto.

2.7 Diagrama de clases

El modelo de diseño permite producir varios modelos del sistema o producto que se va a construir, dicho modelo forma una especie de plan para la solución que será generado. Los diagramas de clases representan un conjunto de interfaces, colaboraciones y sus relaciones. Gráficamente son una colección de nodos y arcos. (20) A continuación se muestra el diagrama de clases de la solución propuesta.

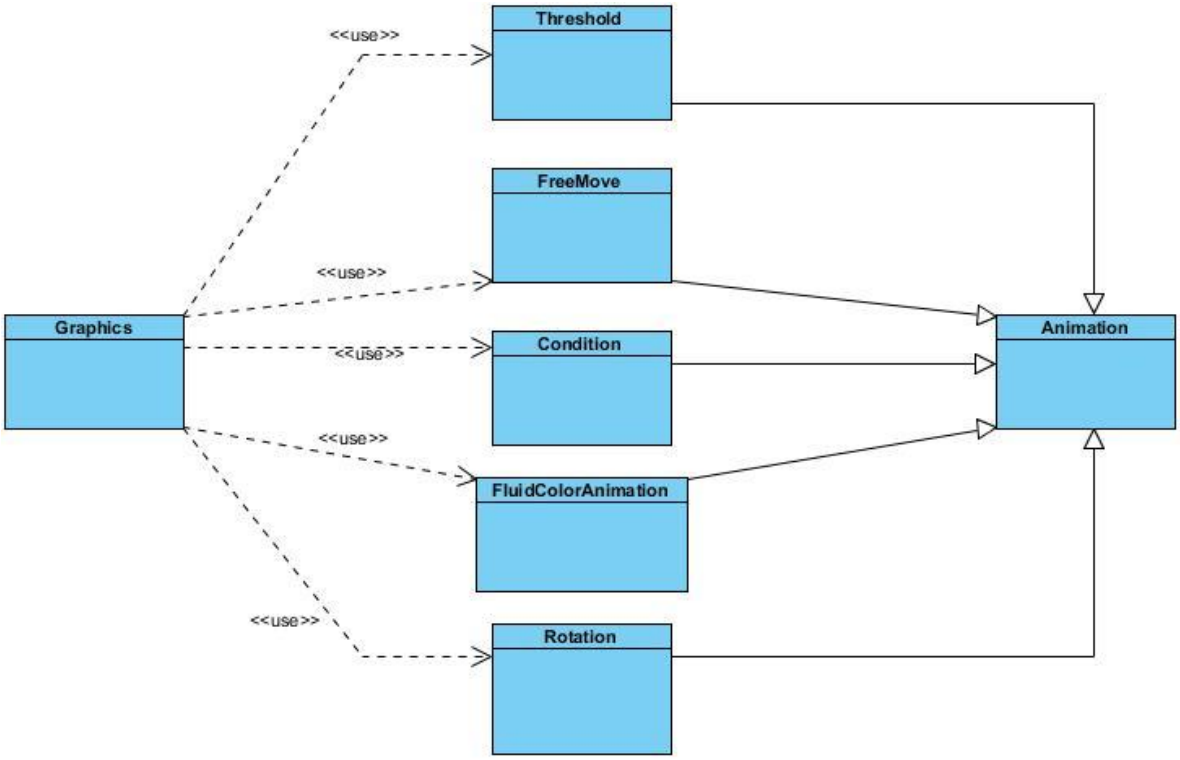


Fig. 6 Diagrama de clases.

2.7.1 Descripción del diagrama de clases

- ✓ ***Graphics:*** Entidad que contiene todos los componentes gráficos.
- ✓ ***Animation:*** Clase que contienes todas las animaciones que se le pueden asociar a un componente gráfico.
- ✓ ***Threshold:*** Clase que admite todos los umbrales asociados a un componente gráfico.
- ✓ ***FreeMove:*** Clase que permitirá darle un movimiento compuesto a un objeto en dependencia del valor de una variable. Esta clase contiene la estructura y métodos para la creación de la animación de movimiento compuesto.

- ✓ ***Condition*:** Clase que va a contener la condición por la cual se va realizar la animación al componente gráfico.
- ✓ ***FluidColorAnimation:*** Entidad encargada de realizar la animación de fluido.
- ✓ ***Rotation:*** Entidad encargada de realizar la animación de fluido.

2.8 Patrón Arquitectónico

Al hacer uso del *framework Qt* para el desarrollo de la aplicación se hereda la arquitectura Modelo-Vista, empleada por el mismo para el manejo de los datos.

Para el desarrollo de la solución se decidió utilizar la arquitectura Modelo-Vista dada la posibilidad que brinda *Qt* de permitir la fácil manipulación de los datos.

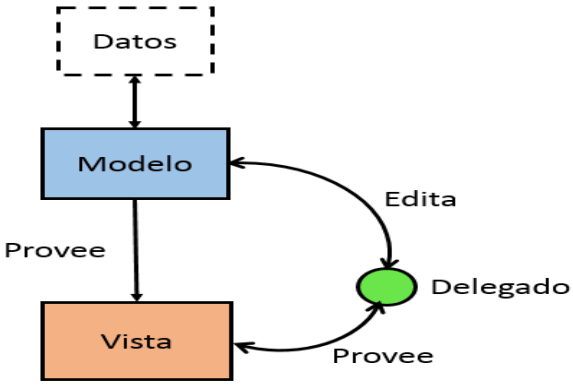


Fig. 7 Patrón de Arquitectura de Software Modelo-Vista.

El modelo se comunica con una fuente de datos, proporcionando una interfaz para los otros componentes en la arquitectura. La naturaleza de la comunicación depende del tipo de fuente de datos, y la forma en que se implementa el modelo. La vista obtiene índices de modelo; estos son referencias a objetos de datos. Mediante el suministro de los índices del modelo, la vista puede recuperar los elementos de datos del origen de datos.

En las vistas estándares, un delegado transforma los datos de manera que puedan ser interpretados de una mejor forma por los usuarios; cuando este objeto se edita, el delegado se comunica directamente con el modelo utilizando sus índices.

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. Las clases abstractas están destinadas a ser una subclase con el fin de proporcionar el conjunto completo de funcionalidad esperada por otros componentes. (19)

Entre las clases modelo utilizadas en la solución se encuentra *ConditionModel* del mismo modo las entidades *GraphicsViewFreeMove* y *SceneFreeMove* entidades que representan la vista.

2.9 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. (16)

2.9.1 Patrones GRASP

Los patrones GRASP (*General Responsibility Assignment Software Patterns*, por sus siglas en inglés) o Patrones Generales de Software para Asignación de Responsabilidades, indican cual es la manera de asignar responsabilidades a objetos software. (16)

Entre los patrones GRASP utilizados en la definición del sistema se encuentran:

- ✓ **Creador:** El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos. En el diseño del sistema se encuentra la clase Animation, la cual se encarga de crear las animaciones que se asocian a los componentes gráficos. En esta relación de composición se evidencia el patrón Creador, pues es la clase *Animation* la responsable de la creación de la clase *FreeMove*.
- ✓ **Controlador:** Permite asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase. Donde un evento del sistema es un evento de alto nivel generado por un actor externo. Este patrón se encuentra evidenciado en la clase *Animation* ya que asigna los permisos a la animación para que esta se inicie o se detenga.
- ✓ **Experto:** Asigna una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. De forma general el diseño del sistema se basa en asignar a cada clase la responsabilidad que solo ellas pueden realizar, pues cada una cuenta con la información necesaria para llevarlas a cabo. Por lo que todas las clases del sistema son expertas.
- ✓ **Bajo acoplamiento:** El uso de los patrones Experto, Creador y Controlador contribuyen al bajo acoplamiento entre las clases del sistema. Este patrón se tuvo presente debido a la importancia que se le atribuye a realizar un diseño de clases

independientes, que puedan soportar los cambios de una manera fácil y que a su vez permitan la reutilización.

- ✓ **Alta cohesión:** Se diseñaron las clases de forma tal que contengan las mínimas responsabilidades necesarias y colaboren con otras para llevar a cabo una tarea. Este patrón permitirá tener clases fáciles de mantener, entender y reutilizar.

2.9.2 Patrones GOF

Los patrones Gang-of-Four (“pandilla de los cuatro”) o comúnmente llamados Patrones GOF, descritos en el libro *Design Patterns*. (20)

Según el libro GOF existen 3 tipos de patrones:

- ✓ De Creación: abstraen el proceso de creación de instancias.
- ✓ Estructurales: se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.
- ✓ De Comportamiento: atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

Para la construcción de la solución, con el objetivo de garantizar una buena práctica de programación, se emplearon los patrones GOF de comportamiento: observador y el método plantilla. A continuación, se presenta una breve descripción de cómo se definen estos patrones y su comportamiento.

- ✓ **Observador:** La mayoría de librerías para el desarrollo de aplicaciones con GUI como Qt, está diseñado para ser dirigidos por eventos, de ahí el famoso sistema de ranuras y señales (*Signals and Slots*) de Qt que no es más que un patrón de diseño observador bastante curado. Señales y ranuras es una construcción del lenguaje introducido en Qt, lo que hace que sea fácil de implementar el patrón observador, evitando código repetitivo.

  Es un patrón de comportamiento que define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambia de estado se notifican y actualizan automáticamente todos los objetos con los cuales se relacionan. El uso de este patrón se ve reflejado en la funcionalidad *Condition*, la cual es utilizada para notificarle al componente si cumple con la condición para que efectúe la animación, haciendo uso de la clase *ConditionValue* que reimplementa la funcionalidad para comparar una instancia de esta clase con otra pasada por parámetros.
- ✓ **Método Plantilla:** Este sencillo patrón resulta útil en casos en los que podamos implementar en una clase abstracta el código común que será usado por las clases

que heredan de ella, permitiéndoles que implementen el comportamiento que varía mediante la reescritura (total o parcial) de determinados métodos. Este patrón se ve reflejado en los métodos *paintRuntime* (), *paintEdition* ().

Conclusiones Parciales

- ✓ En este capítulo se utilizaron varias herramientas para facilitar el proceso de diseño de la aplicación y la definición de sus funcionalidades y características. Entre los artefactos generados por las mismas se encuentran, el modelo conceptual para entender mejor el contexto de la solución, las historias de usuario que permiten una mayor comprensión de los requisitos funcionales del sistema y el diagrama de clases para describir a un nivel más detallado el desarrollo del subsistema.
- ✓ Se hizo uso de los patrones de diseño para dar solución a problemas típicos y recurrentes que se encuentran a la hora de desarrollar una aplicación.

Capítulo 3. Implementación y Pruebas

3.1 Introducción

Una vez diseñada la animación de movimiento compuesto que permita variar en los ejes de coordenadas (x, y) la posición de los componentes gráficos es necesario la implementación de la solución. Posteriormente serán realizadas las pruebas, pues a pesar del correcto funcionamiento, pueden presentar fallas. Es por ello que este capítulo está centrado en la implementación y pruebas, donde será desarrollada y validada la solución propuesta.

3.2 Modelo de implementación

Los diagramas de componentes describen la descomposición física de los elementos de un sistema (módulo, base de datos, programa ejecutable) y sus relaciones. Fundamentalmente, se describe la relación que existe desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos. (21)

3.2.1 Diagrama de componentes

Un diagrama de componentes representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes. Los componentes físicos incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables, o paquetes. Los diagramas de componentes prevalecen en el campo de la arquitectura de *software*, pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. (21)

El diagrama de componente que se presenta a continuación está compuesto por bibliotecas (Library) y ejecutables (Executable):

- ✓ *Graphics*: Biblioteca donde se encuentran agrupadas las entidades encargadas de representar los componentes gráficos que serán utilizados en la representación.
- ✓ *Behavior:* Biblioteca que contiene la biblioteca de animaciones del sistema.
- ✓ *Animation:* Biblioteca donde se encuentran las entidades referentes a las animaciones de los componentes gráficos.

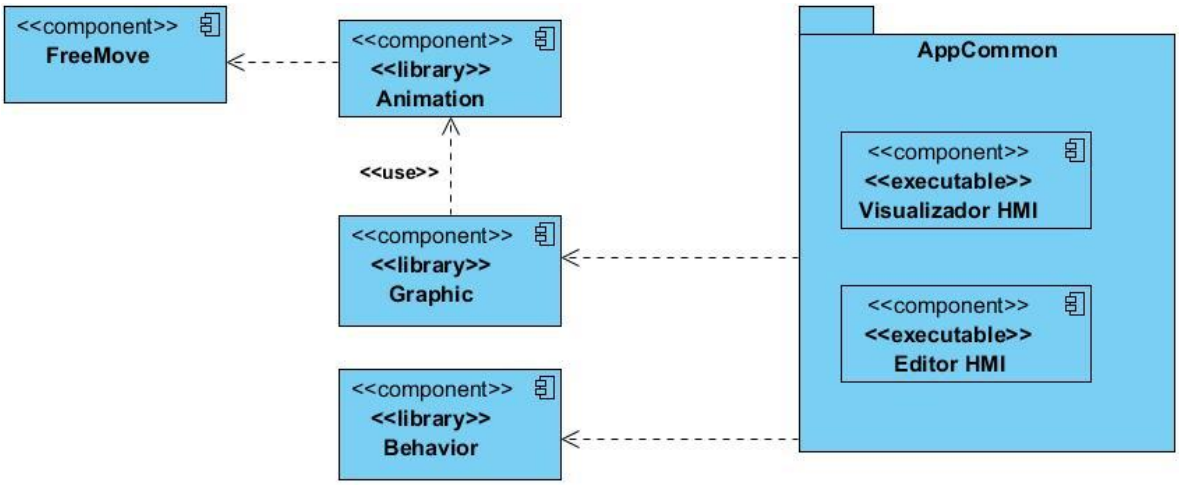


Fig. 8 Diagrama de Componentes.

El sistema general cuenta con elementos comunes para los ambientes de ejecución, el Editor HMI y Visualizador HMI. Estos elementos comunes se dividen en dos bibliotecas especializadas Behavior y Graphics, en el presente desarrollo se agrega a la biblioteca Animation el componente FreeMove para permitir que un componente gráfico realice la animación de movimiento compuesto a partir de la trayectoria configurada.

3.3 Estándar de codificación

Un estándar de codificación son pautas de programación que no están enfocadas a la lógica del programa, sino a su estructura y apariencia física para facilitar la lectura, comprensión y mantenimiento del código (22).

Como la solución propuesta en este trabajo es parte del sistema SCADA SAINUX el estándar de codificación utilizado fue definido por el proyecto:

- ✓ Los atributos de las clases deben comenzar con la letra m seguido de guion bajo y a continuación el nombre del atributo, si existen atributos compuestos la segunda palabra debe comenzar con mayúscula.

  Ejemplo:

```
qreal m_timeBegin;
/**
 * @brief m_timeEnd
 * Variable para definir la hora de finalizacion
 *
 * @autor Keiger Rivera Acosta  krivera@estudiantes.uci.cu
 */
```

- ✓ Ninguna función debe tener más de 200 líneas de código.
- ✓ Las secciones *public*, *protected* y *private* serán declaradas en este orden.
- ✓ Para hacer una descripción breve se adopta el uso del comando @brief. La acción de este comando termina al final de un párrafo, de tal manera que la descripción detallada sigue después de una línea vacía.
- ✓ Para especificar el nombre del autor y la fecha de creación se utilizan los comandos @author y @date.
- ✓ El formato a utilizar a la hora de documentar el código hará que *Doxygen* genere una salida de documentación

A continuación, se presentan dos ejemplos del estándar de codificación usado:

```
/**
  * @brief calculatePos
  * metodo para calcular la posicion exacta de unj tiempo pasado por parametro
  *
  * @param _timeCurrent
  * Tiempo dado en una trayectoria
  *
  * @return tPointF QPointF
  *
  * @autor Keiger Rivera Acosta  krivera@estudiantes.uci.cu
  */
```

```
CIM::DataType::DataTypes::CollectionsPointF listPoint();
/**
  * @brief CollectionsPointF
  *
  * Metodo que devuelve la lista de puntos que el usuario sleccione en la escena
  *
  * @autor Keiger Rivera Acosta  krivera@estudiantes.uci.cu
  *
  * @date febrero 23, 2017
  */
```

- ✓ Todo el código debe de pertenecer a los espacios de nombre definidos por la empresa y deben cumplir la estructura:

  EMPRESA::PRODUCTO::Módulo

  Ejemplo.

```
namespace ESCMGA
{
namespace Behavior
{
namespace Animation
{
```

Ejemplo:

```
} // namespace Animation
} // namespace Behavior
} // namespace ESCMGA


Q_DECLARE_METATYPE(ESCMGA::Behavior::Animation::FreeMove*)
Q_DECLARE_METATYPE(ESCMGA::Behavior::Animation::FreeMove::ModeMove)

#endif /*FREEMOVE_H*/
```

3.4 Implementación de la Trayectoria del Movimiento Compuesto

Para el diseño de la trayectoria se usó la biblioteca *QGraphicsView* dentro de *Qt* que permite la creación e iteración de elementos gráficos y que usa el método de programación Modelo/Vista. De manera simple, varias vistas pueden observar una misma escena y una escena puede contener elementos gráficos de diferentes formas geométricas.

El framework cuenta con 3 elementos: *QGraphicsScene* (La escena), *QGraphicsItem* (El ítem), *QGraphicsView* (La Vista) para crear la escena, un ítem y luego visualizarlo.

3.4.1 Escena de los Gráficos

Representa una escena con elementos. Es la clase que se encarga de almacenar los *widgets*. Para el desarrollo de la solución se creó la clase *SceneFreeMove* la cual hereda de la clase QGraphicsScene para la creación de la escena del Movimiento Compuesto. Dentro de la clase *SceneFreeMove* encuentra como miembro de la clase el método *listPoint*.

El fragmento de código que se encuentran escrito en el lenguaje de programación c++, del método *listPoint m*uestra cómo se obtiene la lista de puntos que el usuario selecciona para el trayecto, devuelve la lista de puntos para definir la trayectoria, luego convierte cada punto de la escena a mapa pixeles para realizar la animación de un componente gráfico utilizando el mismo trayecto. A continuación, se muestra el fragmento de código.

```
CIM::DataType::DataTypes::CollectionsPointF
 FreeMove::listPoint()
{

      if(QGraphicsItem * item=dynamic_cast<QGraphicsItem*>(object))
      {
         m_startingPositionObject=item->pos();
         m_sceneRect=item->scene()->sceneRect();
      }
   }
```

```
tPoints.push_back(QPointF(m_sceneRect.x(),m_sceneRect.y()));
   tPoints.push_back(QPointF(m_sceneRect.width(),m_sceneRect.height()));
   tPoints.push_back (QPointF (-1,-1));
   tPoints.push_back(m_startingPositionObject);
   tPoints.push_back (QPointF (-1,-1));

   int tSize=m_listPoint.size();

   for (int i=0; i<tSize;i++)
      tPoints.push_back(m_listPoint.at(i));

   QPixmap tPixmapScene=convertSceneToPixmap();

   return tPoints;
```

También como miembro de la clase se encuentra el método *markSelectedPoint*(*QPointF_pos*), dada una posición dentro de la escena que es pasada por parámetro se busca cuáles de los puntos que integran el trayecto están en esa posición o cerca de su rango de proximidad, en caso de existir varios se marca el primero, este método garantiza que todos los puntos de la trayectoria pertenezcan a la misma. La posición del punto seleccionado queda registrada en el atributo de la clase *m_indexPointSelected*, así se garantiza que todos los puntos del trayecto pertenezcan a la escena. A continuación, se muestra el fragmento de código.

```
void SceneFreeMove::markSelectedPoint(QPointF _pos)
{
   int tSize=m_points.size();

   QPointF tPoint;
   m_indexPointSelected=-1;

   for(int i=1;i<tSize && m_indexPointSelected==(-1);i++)
   {
      tPoint=m_points.at(i);

      if(abs(_pos.x()-tPoint.x())<=DIAMETER &&
            abs(_pos.y()-tPoint.y())<=DIAMETER)
      {
         m_indexPointSelected=i;
      }}}
```

3.4.2 Elementos gráficos

Representa un grupo de *items*. Es una clase para el manejo de elementos gráficos en la escena y proporciona varios *items* estándar para formas como rectángulos, elipses y textos.

También soporta eventos del mouse como mover, soltar, presionar, doble *click*, y eventos del teclado.

QGraphicsItem, además soporta dos características importantes en elementos gráficos: *Drag* and *Drop* (Soltar y arrastrar). Cada objeto QGraphicsItem en la escena soporta rotación, *zooming* y traslado. El método *clickedBttons* devuelve cuales son los botones de clic. A continuación, se muestra como fue utilizado el código para el desarrollo de la solución.

```
void PathSettingsFreeMove::clickedBttons(QAbstractButton *button)
{
   QDialogButtonBox::ButtonRole role = m_buttonBox->buttonRole(button);

   if (role == QDialogButtonBox::RejectRole)
   {
      reject ();
   }
   else
   {
      accept ();
   }
}
```

3.4.3 Vista Gráfica

La clase que se encarga de proporcionar los *widgets* que visualizan el contenido de una escena. La vista recibe eventos de entrada del teclado o del mouse, y los traslada a la escena. Para agregar elementos a una escena empieza construyendo un objeto QGraphicsScene. A continuación, se tienen dos opciones: agregar los objetos *QGraphicsItem* existentes llamando a *addItem*, o llamar a una de las funciones de conveniencia *addEllipse*, *addLine*, *addPath*, *addPixmap*, *addPolygon*, *addRect*, O *addText*, que todos devuelven un puntero al elemento recién agregado. Las dimensiones de los elementos añadidos con estas funciones son relativas al sistema de coordenadas del elemento y la posición de los elementos se inicializa en (0, 0) en la escena.

3.4.4 Configuración de la ruta

Para el ajuste de la ruta se definió la clase *PathSettingsFreeMove* que va a contener el método *PathSettingsFreeMove.* Este método se encarga de ajustar la ruta de movimiento compuesto que realizara un componente, haciendo uso de la lista de puntos obtenida del método *CollectionsPointFtList* esta lista va hacer de tipo de dato *QPointF. QPointF* es una clase que define un punto en el plano usando precisión de punto flotante para devolver la posición del cursor del ratón en las coordenadas de la escena donde se hizo clic.

Un punto se especifica mediante una coordenada X y una coordenada Y a la que se puede acceder mediante las funciones X e Y. Las coordenadas del punto se especifican usando números de coma flotante para la exactitud. La función *isNull* devuelve verdadero si tanto X como Y están establecidos en (0.0). Las coordenadas pueden ser ajustadas (o alteradas) usando las funciones setX y setY, o alternativamente las funciones rx y ry que devuelven referencias a las coordenadas (permitiendo la manipulación directa). Para la conexión de los puntos definidos para la trayectoria se definió el método *createConnects.*

Para crear el espacio de trabajo para la configuración del trayecto de la animación se definió el método *createWorkSpace* () perteneciente a la clase *PathSettingsFreeMove* a continuación se muestra el fragmento de código que implemento para realizar la acción.

```
void PathSettingsFreeMove::createWorkSpace()
{
    m_gridLayout = new QGridLayout(this);
    m_graphicsView = new GraphicsViewFreeMove(this);
    m_scene=new SceneFreeMove(m_graphicsView);
    m_graphicsView->setScene(m_scene);
    m_gridLayout->addWidget(m_graphicsView, 0, 0, 1, 1);

    m_horizontalLayout = new QHBoxLayout();
    m_horizontalSpacer = new QSpacerItem(328, 20, QSizePolicy::Expanding,
QSizePolicy::Minimum);

    m_horizontalLayout->addItem(m_horizontalSpacer);
    m_buttonBox = new QDialogButtonBox(this);
    m_buttonBox->setOrientation(Qt::Horizontal);
    m_buttonBox->setStandardButtons(QDialogButtonBox::Cancel|QDialogButtonBox::Ok);
    m_horizontalLayout->addWidget(m_buttonBox);

    m_gridLayout->addLayout(m_horizontalLayout, 1, 0, 1, 1);
    setWindowTitle("Configurar trayecto de la animación");
    setWindowIcon(RESOURCE_THEME_ICON("Sainux_icono.png"));
```

FreeMove

Para permitir que un elemento grafico realice el movimiento compuesto se definió el método *FreeMove* que pertenece a la clase *FreeMove,* esta clase va a contener todo lo referente a la animación de movimiento compuesto.

3.5 Configuración de la trayectoria del Movimiento Compuesto

3.5.1 Inspector de propiedades del componente gráfico

Para lograr la visualización de la animación de movimiento compuesto hay que asociarle la animación a un componente gráfico, se selecciona el componente gráfico, seleccionar la opción Propiedades, de esta forma se muestra el inspector de propiedades donde elige la opción animaciones y se asocia la animación al componente. La figura muestra un despliegue del SCADA SAINUX y señalado en rojo el inspector de propiedades del componente gráficos al cual se le va asociar la animación de movimiento compuesto.

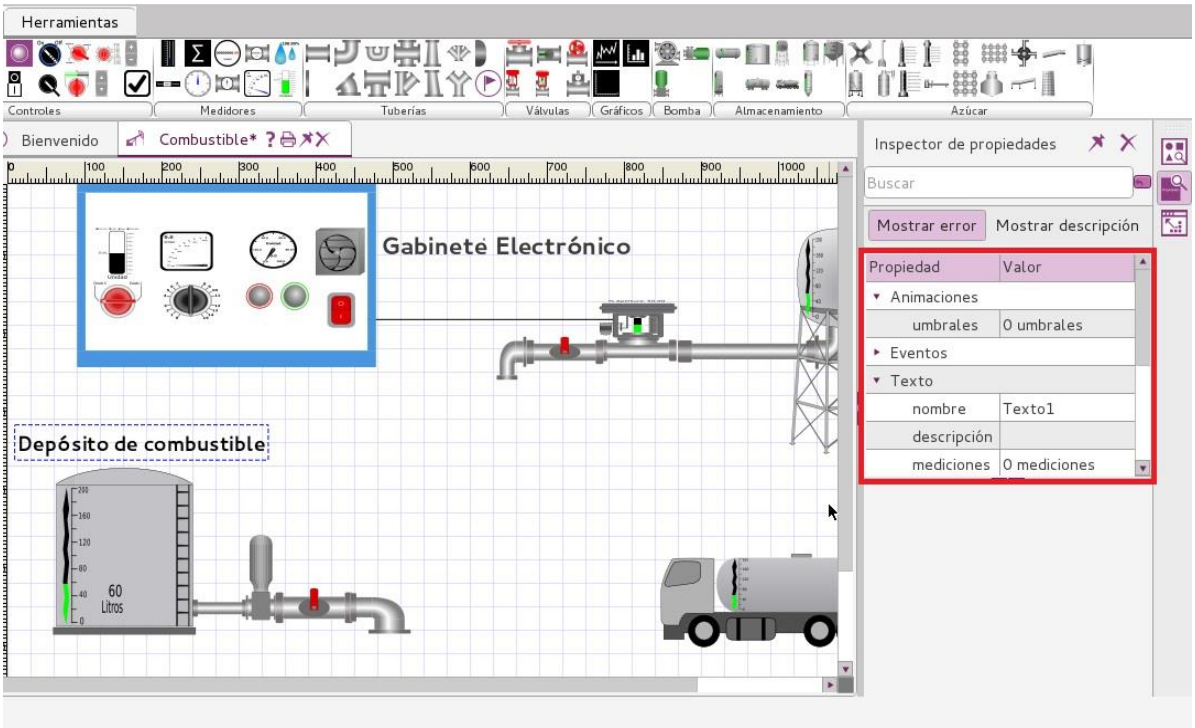


Fig.9 Despliegue del SCADA SAINUX

3.5.2 Animación de movimiento compuesto y parámetros de configuración

La figura muestra la colección de animaciones que permite el componente gráfico y parámetros de configuración de la animación de movimiento compuesto.

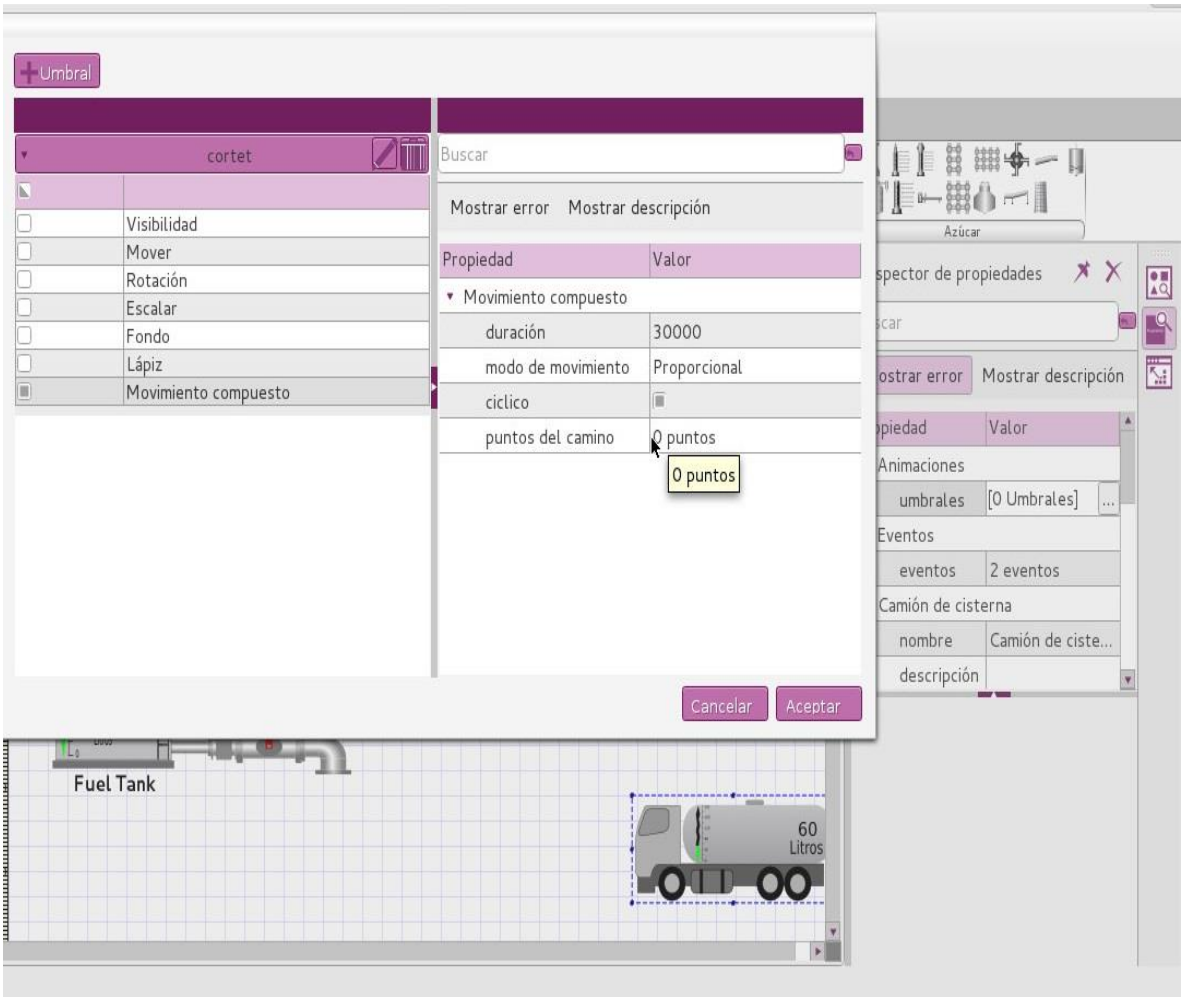


Fig.10 Animación de movimiento compuesto y parámetros de configuración.

3.5.3 Ambiente de edición para crear la trayectoria del movimiento

Para lograr la visualización de la animación de movimiento compuesto hay que configurarle la animación a un componente gráfico. La configuración de la trayectoria va a permitir una edición que facilite trazar la trayectoria que seguirá el componente en proporción al valor de la variable asociada, para un valor inicial de la variable el objeto se encuentre en la posición inicial, para un valor final de la variable el objeto se encuentre en una posición final y para valores intermedios se encuentre en una posición dentro de la trayectoria determinada, después de haber hecho la configuración del movimiento se espera que el

componente realice la trayectoria configurada, en la figura se muestra el ambiente de edición para crear la trayectoria del movimiento.

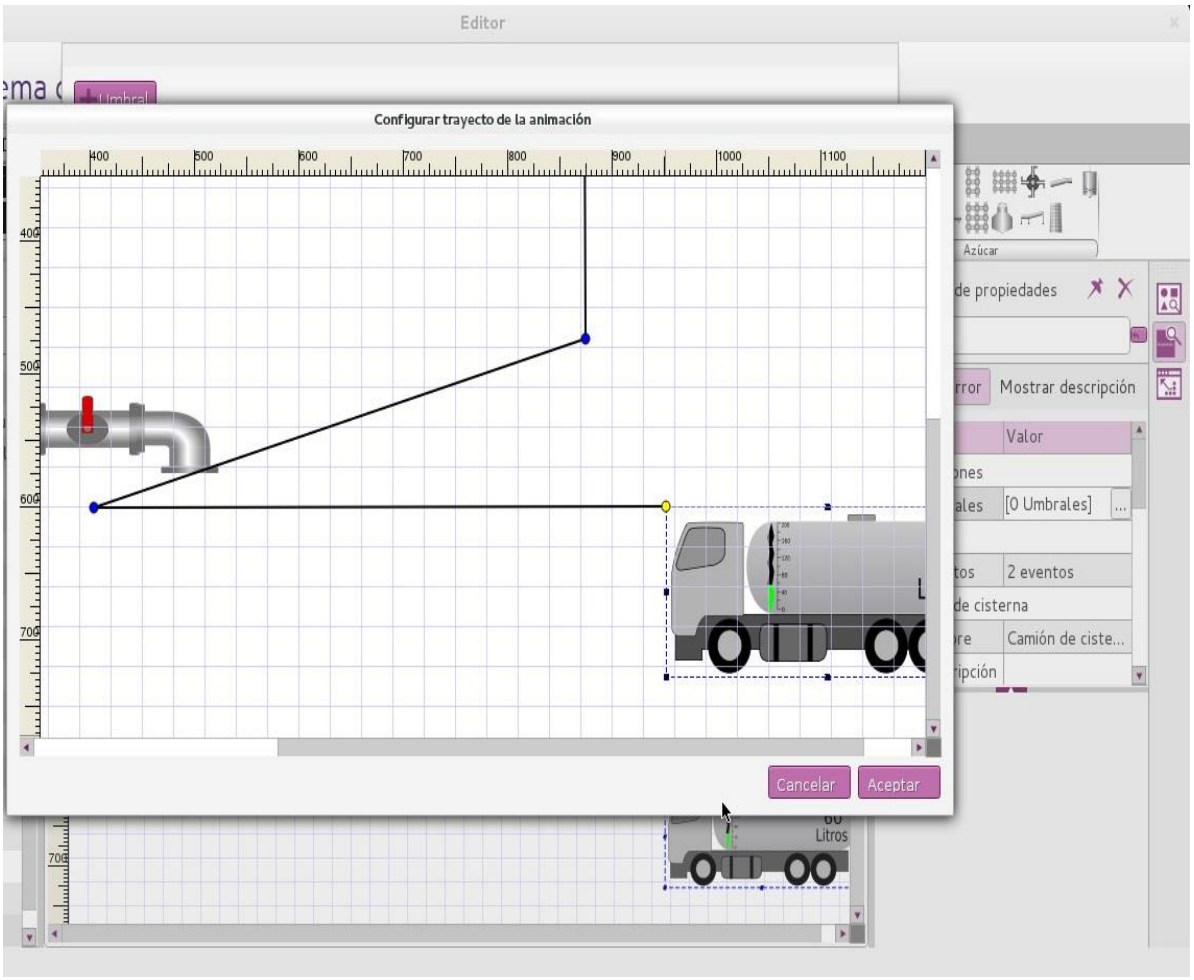


Fig.11 Ambiente de edición de la trayectoria del componente.

3.6 Pruebas de software

El instrumento adecuado para determinar la calidad de un producto software es el proceso de pruebas. Las pruebas de software representan el porcentaje más grande de esfuerzo técnico en el proceso de software. Sin importar el tipo de software que se construya, una estrategia para planificar, ejecutar y controlar pruebas sistemáticas comienza por considerar pequeños elementos del software y moverse hacia afuera, hacia el programa como un todo (19).Se selecciona para comprobar la calidad de la solución desarrollada las pruebas de aceptación, ya que este es el tipo de pruebas recomendado por la metodología de desarrollo empleada. Es válido aclarar que estas pruebas las realiza el propio cliente acompañado del equipo de desarrollo y se orientan a las funcionalidades del sistema.

Para la realización de las pruebas de aceptación se empleó la técnica Partición de Equivalencia. Esta permite examinar los valores válidos e inválidos de las entradas existentes en el software (24).

3.6.1 Diseño de casos de prueba

Un caso de prueba cubre el software más a fondo y con más detalle que un caso de uso. Los mismos incluyen todas las funciones que el programa es capaz de realizar. Estos deben tener en cuenta el uso de todo tipo de datos de entrada/salida, cada comportamiento esperado, todos los elementos de diseño y cada clase de defecto. Todos los requisitos deberán ser cubiertos por los casos de prueba (25).

A continuación, se muestran los CP correspondientes a las pruebas de aceptación realizadas.

Prueba de Aceptación	
Número: 1	**Historia de usuario: 1**
Nombre: Asociar la animación a los componentes gráficos	
Condiciones de Ejecución: Debe estar creado un despliegue	
Entradas/ Pasos de Ejecución: ✓ El usuario seleccionar componente. ✓ El usuario da Clic derecho con el ratón y abre inspector de propiedades. ✓ El usuario accede a la opción animaciones. ✓ El usuario selecciona la opción umbrales y se le asocia la animación de movimiento compuesto al componente.	
Resultado esperado: La animación se le asocia al componente gráfico.	
Evaluación de la prueba: Se realiza la primera iteración donde fueron identificadas varias no conformidades, se solucionan estas y se realiza otra iteración que arrojó una evaluación satisfactoria.	

Tabla 5. Caso de prueba de Aceptación 1.

Prueba de Aceptación	
Número: 2	**Historia de usuario: 2**
Nombre: Definir la trayectoria del componente gráfico.	
Condiciones de Ejecución: Debe estar visualizado el componente grafico en el ambiente de edición.	
Entradas/ Pasos de Ejecución: ✓ El usuario selecciona la animación de movimiento compuesto en la colección de umbrales. ✓ El usuario accede a la opción propiedad del movimiento. ✓ El usuario selecciona la opción puntos del camino. ✓ El usuario edita la trayectoria del componente gráfico.	
Resultado esperado: Se muestra el ambiente de edición para configurar la trayectoria del componente.	
Evaluación de la prueba: Se realiza una primera iteración donde se encontraron varias no conformidades, se corrigen estas, al realizar una segunda iteración se detectan otras no conformidades que son corregidas y en una tercera iteración se obtiene un resultado de satisfactorio.	

Tabla 6. Caso de prueba de Aceptación 2.

Prueba de Aceptación	
Número: 3	**Historia de usuario: 3**
Nombre: Eliminar punto de la trayectoria	
Condiciones de Ejecución: Debe estar visualizada la trayectoria la trayectoria en el ambiente de edición así como los puntos del trayecto.	
Entradas/ Pasos de Ejecución: ✓ El usuario selecciona el punto que desea eliminar. ✓ El usuario da clic derecho sobre el punto y selecciona la opción eliminar punto.	

Resultado esperado: Se elimina el punto de la trayectoria.
Evaluación de la prueba: Se realiza una primera iteración donde se encontraron varias no conformidades, se corrigen estas, al realizar una segunda iteración se detectan otras no conformidades que son corregidas y en una tercera iteración se obtiene un resultado de satisfactorio.

Tabla 7. Caso de prueba de Aceptación 3.

Prueba de Aceptación	
Número: 4	**Historia de usuario: 4**
Nombre: Ejecutar la animación en el ambiente de visualización.	
Condiciones de Ejecución: Debe mostrarse el ambiente de Visualización.	
Entradas/ Pasos de Ejecución: ✓ El usuario selecciona el despliegue creado en el ambiente de configuración. ✓ El usuario visualiza la trayectoria del componente.	
Resultado esperado: Se ejecuta la animación en el ambiente de visualización.	
Evaluación de la prueba: Se realiza una primera iteración donde se encontraron varias no conformidades, se corrigen estas, al realizar una segunda iteración se detectan otras no conformidades que son corregidas y en una tercera iteración se obtiene un resultado de satisfactorio.	

Tabla 8. Caso de prueba de Aceptación 4.

3.7 Desarrollo de las iteraciones de pruebas

Durante el plan de iteraciones realizado, para llevar a cabo la fase de pruebas con el objetivo de validar el correcto funcionamiento de la aplicación desarrollada se encontraron varias no conformidades. El diseño de prueba definido se ejecutó sobre el subsistema de animación de movimiento compuesto para definir la trayectoria del componente graficó en busca de no conformidades sobre las funcionalidades del sistema. En cada iteración se probaron todas las funcionalidades, desarrollándose 3 de ellas, cuyos resultados se presentan a continuación:

Iteración #1

Durante las pruebas en esta iteración se encontraron 10 no conformidades, las cuales fueron resueltas inmediatamente.

Iteración #2

Durante las pruebas en esta iteración se encontraron 8 no conformidades, las cuales fueron resueltas inmediatamente.

Iteración #3

Durante las pruebas en esta iteración no se encontraron no conformidades.

3.8 Clasificación de las no conformidades

Se encontraron un total de 18 no conformidades: 7 errores validación, 5 errores de inicialización y 6 de visualización.

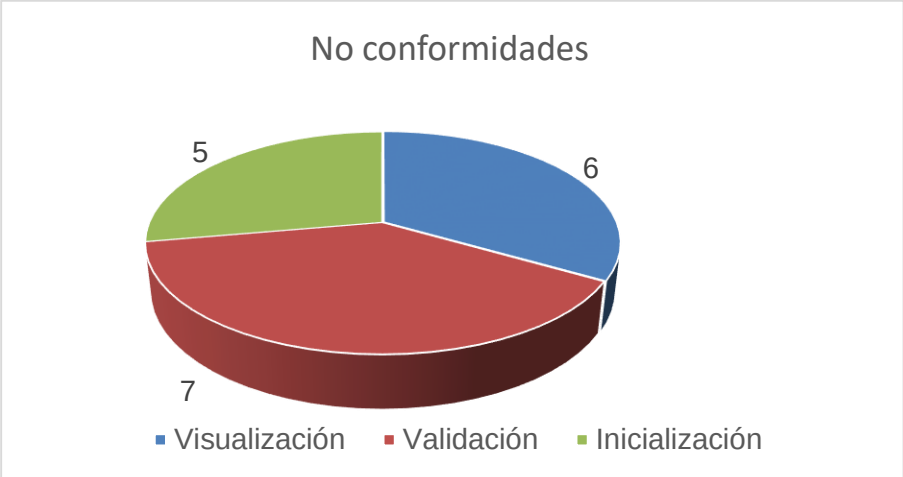


Fig. 12 Clasificación de las no conformidades.

Conclusiones Parciales

- ✓ En el desarrollo del capítulo se presentó la distribución física y lógica del sistema y sus componentes mediante el diagrama de componentes.

- ✓ Finalmente, se valida el subsistema desarrollado mediante pruebas de caja negra, realizando el caso de prueba correspondiente. Este proceso permitió detectar las no conformidades que presentaba el subsistema y dar solución a las mismas.

Conclusiones generales

Una vez concluida la presente investigación se concluye que:

- ✓ La animación de movimiento compuesto les permite a los sistemas SCADA cubrir aquellos escenarios de automatización donde se necesiten representaciones gráficas que durante su movimiento varíen ambos ejes de coordenadas en un plano cartesiano.
- ✓ La biblioteca de animaciones del SCADA SAINUX se enriquece con la incorporación de la animación de movimiento compuesto, permitiendo añadir mayor versatilidad y utilidad a los componentes gráficos existentes y a los nuevos que se incorporen al sistema.

- ✓ La animación de movimiento compuesto propuesta en el presente trabajo satisface las necesidades de variar ambas coordenadas de posición de los componentes gráficos del HMI del SCADA SAINUX, siendo validada su implementación mediante la aplicación de pruebas de caja negra, dándosele así cumplimiento a los objetivos trazados para la presente investigación.

Recomendaciones

Teniendo en cuenta los resultados obtenidos en esta investigación y basado en la experiencia adquirida, se recomienda:

- ✓ Implementar un mecanismo que permita exportar e importar entre componentes similares las trayectorias de las animaciones de movimiento compuesto configuradas.
- ✓ Profundizar en el estudio de las animaciones de movimiento compuesto en sistemas SCADA con el objetivo de extender las funcionalidades de las animaciones de movimiento compuesto implementada en el SCADA SAINUX.

Glosario de Términos

Despliegue: Es un resumen esquematizado con la ventaja de permitir visualizar la estructura y organización de los elementos que componen el proceso.

Framework: En el desarrollo de software, un framework es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

UML: Lenguaje Unificado de Modelado o UML (del inglés *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (*Object Management Group*). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes reutilizables.

SVG: Por sus siglas en inglés (Scalable Vector Graphics) es un formato de imagen vectorial basado en XML para gráficos bidimensionales con soporte para interactividad y animación. La especificación SVG es un estándar abierto desarrollado por el World Wide Web Consortium (W3C) desde 1999.

Procesos de campo: Actividades que implican la participación de un número determinado de elementos gráficos u otros artefactos en un SCADA.

Dispositivos de campo: Conjuntos de elementos que conforman un SCADA

Referencias Bibliográficas

1. Industrial, CEDIN Informática. gespro. Suite de Gestión de Proyectos. [En línea] 20 de 4 de 2015. [Citado el: 31 de 5 de 2017.] http://gespro.cedin.prod.uci.cu.

2. Crespo, Wilian. WordPress.com. [En línea] 9 de 2 de 2011. [Citado el: 31 de 5 de 2017.] https://automatizacionindustrial.wordpress.com.

3. Pérez López, Esteban. Los sistemas SCADA en la automatización industrial. Costa Rica : s.n., 2015. Vol. 28.

4. Gabrie Lordonez. Funciones de Sistemas SCADA. 2009.

5. Creación Digital. [En línea] 28 de 01 de 2013. [Citado el: 19 de 11 de 2016.] http://creaciodigital.upf.edu/~smiguel/b12animacion.html.

6. Pi. Software de Adquisión, supervisión y control: una evolución permanente. Automática e Instrumentación. 2003.

7. InduSoft. InduSoft Web Studio. EE.UU. : s.n., 2012. pág. 93.

8. Pressman., Roger S. Ingeniería del Software. Un enfoque Práctico. [En línea] 3 de 2 de 2016. [Citado el: 5 de 3 de 2017.] http://bibliodoc.uci.cu/pdf/8448111869.pdf..

9. MEJORA, PROGRAMA DE. Metodología de Desarrollo para la Actividad Productiva de la UCI. [En línea] Tamara Rodríguez Sánchez. [Citado el: 30 de 11 de 2016.] http://excriba.prod.uci.cu/page/context/shared/sharedfiles/Metodologiauci.pdf.

10. CORPORATION, NOKIA. Qt. [En línea] 2008. [Citado el: 2 de diciembre de 2015.] http://qt.nokia.com/products/developer-tools/.

11. Company, Qt. [En línea] 2016. [Citado el: 8 de 6 de 2017.] http://blog.qt.io/blog/category/qtcreator/.

12. Rumbaugh, James, Jacobson, Ivar y Booch, Grady. El lenguaje unificado de modelado. s.l. : Adison-Wesley, 2000.

13. CMM. CMM Benchmark.net. [En línea] [Citado el: 8 de 6 de 2017.] http://es.ccm.net/download/descargar-28127-visual-paradigm-for-uml-enterprise-edition.

14. Subversion. The Apache Software Foundation. [En línea] [Citado el: 8 de 6 de 2017.] https://subversion.apache.org/.

15. Larman, Craig . UML y Patrones MODELO DEL DOMINIO. s.l. : Prentice Hall, 2003.

16. Pearson. Sommerville, Ingeniería del Software. Vol. 7. 2005.

17. Cardozzo., Daniel Ramos. Desarrollo de Sotware. Campus Academy. 2014.

18. S.Pressman, Roger. INGENIERÍA DEL SOFTWARE. UN ENFOQUE PRÁCTICO. México, D. F. : McGRAW-HILL INTERAMERICANA EDITORES, S.A. DE C.V., 2010.

19. Microsoft. Diagramas de clases de UML: Instrucciones. [En línea] 2017 . https://msdn.microsoft.com/es-es/library/dd409416.aspx.

20. Gamma, Erich, Helm , Richard y Johnson, Ralph . Design Patterns. [aut. libro] Kevin Zhang. Elements of Reusable Object-Oriented Software. 1996.

21. Espinoza, M.C. José Martín Olguín. Análisis Orientado a Objetos.Ingeniería del Software. 2004.

22. Ordoñez, Rafael Alejandro Pérez. Estándares de codificación para C++. Carretera a San Antonio Km 2 ½ . Torrens. Boyeros. Ciudad de La Habana. Cuba : s.n., 2016.

23. 4rsoluciones. [En línea] 28 de 11 de 2012. [Citado el: 4 de 4 de 2017.] http://www.4rsoluciones.com/blog/test-de-aceptacion-el-ultimo-paso-para-el-aseguramiento-de-calidad-en-software-2/.

24. Definición de caso de prueba. Testeando el Software. [En línea] [Citado el: 13 de mayo de 2016.] http://testeandosoftware.com/casos-de-uso-vs-casos-de-prueba..