Facultad 5

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Título: Mecanismo de incorporación de scripting al HMI del SCADA Guardián del ALBA.

Autor: Yosvani Ramírez Martínez

Tutor: Ing. Roberto Cárdenas Isla

Co-Tutor: Ing. Jorge Carrera Ortega

“La Habana, Mayo de 2013“
“Año 55 de la Revolución”

DECLARACIÓN DE AUTORÍA.

Declaro que soy el único autor de este trabajo y autorizo al Centro de Informática Industrial de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

Yosvani Ramírez Martínez

_____
Firma del autor

Roberto Cárdenas Isla

_____
Firma del tutor

DATOS DE CONTACTO

Tutor: Roberto Cárdenas Isla

Categoría Científica: Ingeniero

Especialidad: Ingeniero en Ciencias Informáticas

Correo Electrónico: **risla@uci.cu**

Categoría docente: Instructor

Años de experiencia: 4

Años de graduado: 4

Tutor: Jorge Carrera Ortega

Categoría Científica: Ingeniero

Especialidad: Ingeniero en Ciencias Informáticas

Correo Electrónico: **jorge.carrera@vcl.desoft.cu**

Categoría docente: -

Años de experiencia: 3

Años de graduado: 2

DEDICATORIA

A mi familia en general por el apoyo incondicional que me han brindado y en especial:

A mis abuelos por todas las enseñanzas que me han dado
y por siempre estar ahí cuando los necesité.

A mis padres por ser la luz que siempre me ha iluminado el camino,
por ayudarme en todo momento y depositar su confianza en mí.

A mi tío Ernesto, a quien siempre he tenido como ejemplo a seguir.

A mis hermanos y en especial a Yunior por estar en las buenas y en las malas.

A todos los que confiaron en mí y de una forma u otra contribuyeron en mi formación.

A mis amigos y a los que no me consideraron así, por darme tantos motivos
para salir adelante y hacer las cosas cada vez mejor.

AGRADECIMIENTOS

Agradezco a la Revolución y al comandante Fidel por darme la oportunidad de estudiar en esta escuela y formarme como profesional.

Quiero agradecer de forma general a mi familia por siempre confiar en mí.

Agradezco a mi tutor, por ser un magnífico guía y considerarlo un amigo más.

A todos los profesores que hicieron de mí una persona de bien, por darme la mejor enseñanza del mundo y formarme como profesional.

A todos mis compañeros de proyecto a los que molesté tanto para aprender la mayoría de las cosas que hoy conozco.

A mis amigos que tanto me apoyaron y me ayudaron a levantar la cabeza cuando la tenía gacha, a Yirenia y Yanira por estar junto a mí durante estos 5 años.

RESUMEN

La informática ha alcanzado un elevado desarrollo en diversas esferas a nivel mundial, entre ellas: la industria. Tal avance ha impuesto un reto, el perfeccionamiento constante y la capacitación de personas que se involucran en impulsar el progreso industrial y dar solución a los problemas que se derivan de esta evolución. Entre los avances antes mencionados se encuentran los sistemas de supervisión, control y adquisición de datos (SCADA), estos permiten supervisar y controlar a distancia una instalación, proceso o sistema de características variadas. En el Centro de Informática Industrial (CEDIN) de la Universidad de las Ciencias Informáticas (UCI), se desarrolla, en convenio con la Empresa de Petróleos de Venezuela S.A. (PDVSA), el SCADA Guardián del ALBA (GALBA). Este SCADA está compuesto por varios módulos tales como: Comunicación, Procesamiento, Interfaz Hombre-Máquina (HMI), Aplicaciones, Seguridad y Base de Datos Históricos.

En algunos sistemas SCADA, se han incorporado lenguajes que permiten programar tareas que respondan a eventos del sistema, personalizando las funcionalidades del módulo HMI, logrando así desarrollar aplicaciones más eficientes y complejas. El presente trabajo se basa en el desarrollo de una aplicación visual, que permita la gestión de una lógica operacional que perfeccione las funcionalidades del HMI del SCADA Guardián del ALBA.

El desarrollo de la aplicación está sustentado por la metodología ágil XP, utilizando las técnicas de modelación establecidas por el Lenguaje Unificado de Modelado (UML) y el potente marco de trabajo para el diseño de interfaces gráficas Qt.

Palabras Clave: HMI, Lógica operacional, Script, Secuencia de comandos, Programación script.

Índice de Contenido

Índice de Ilustraciones

Índice de Tablas

INTRODUCCIÓN

La humanidad ha transitado por varias etapas en su desarrollo, lo cual ha exigido una constante investigación científica que impulse su evolución. Con el trabajo y la experiencia adquirida se han generado constantes cambios, tanto en la ciencia como en la tecnología, demandando nuevos métodos y técnicas eficientes que favorezcan un equilibrio de estos campos a nivel mundial. Actualmente las industrias, en su mayoría, se encuentran automatizadas; aunque generalmente no da abasto el avance tecnológico adquirido, por tal motivo, la automática se ve enfrascada en una batalla de constante innovación.

En las industrias se utilizan diferentes herramientas para la ejecución, monitoreo y control de los procesos industriales; los que al inicio de la automatización, eran bastante sencillos. Recientemente han ganado en complejidad y funcionalidad de forma exponencial, esto ha permitido una disminución considerable en la carga de trabajo de los trabajadores en cuanto a tareas se refiere, así como la peligrosidad en las labores que realizan. En la actualidad, los procesos industriales se monitorean y controlan mediante los sistemas de supervisión, control y adquisición de datos (SCADA, por sus siglas en inglés). Los sistemas mencionados anteriormente, proporcionan gran información de los procesos de forma oportuna para la toma de decisiones y centralizan el funcionamiento de una empresa en una reducida cantidad de estaciones de trabajo, mejorando altamente la eficacia de dichos procesos.

En la Universidad de las Ciencias Informáticas (UCI) se implementa un esquema estructurado de estudio-trabajo, en el cual se instruye a los estudiantes tanto en su formación docente como en su vida laboral, basado en la integración de procesos fundamentales como la formación, investigación y la producción en torno a una temática de un centro de desarrollo. Dentro de la universidad se cuenta con un Centro de Informática Industrial (CEDIN), perteneciente a la facultad 5, el cual desarrolla el software SCADA Guardián del ALBA (GALBA).

El GALBA consta con un módulo Interfaz Hombre Máquina (HMI) que se divide en dos entornos: Entorno de Configuración (EC) y Entorno de Visualización (EV). El EC permite al mantenedor del sistema editar el ambiente de trabajo, configurar la seguridad del sistema, definir los parámetros de variables a supervisar

así como el tipo de datos, alarmas, salva de información, además diseñar los despliegues[1] y componentes gráficos. El EV es el encargado de visualizar la configuración realizada en el EC y permitir al operador interactuar con el sistema, supervisar alarmas y puntos, analizar información a través de tendencias y reportes de estado lo cual brinda una mejor calidad en el control de los procesos.

En ocasiones las tareas de configuración en el EC se tornan complejas para los mantenedores por el alto nivel de detalle del proceso a visualizar en los despliegues y la ausencia de un mecanismo que permita la configuración de una manera más sencilla. Esto provoca que a los mantenedores se le dificulten las tareas de personalizar ciertas propiedades de los gráficos y animaciones utilizados en el diseño; ejemplo de ello es cuando se desea utilizar funciones matemáticas (seno, coseno, logaritmo, etc.) para el cálculo y ejecución de animaciones de movimiento, para definir mejor una secuencia de animaciones, el posicionamiento o distribución de los gráficos en el espacio, etc. Otro elemento importante es la dificultad en el momento de configurarse mediante inspección de propiedades de los gráficos los eventos asociados a estos como son: presionado, liberado, seleccionado, doble selección, entre otros. Además es imposible el vínculo con aplicaciones externas que facilitarían dicho proceso para aquellos que lo realizan. Por lo antes planteado se puede caracterizar hoy la configuración en el HMI como un proceso complejo que limita a los mantenedores de realizar configuraciones más completas y con el alto nivel de detalle que caracteriza a un SCADA actual.

A partir de la **situación problémica** planteada anteriormente, se propone como **problema de investigación** el siguiente: ¿Cómo contribuir al proceso de configuración en el módulo HMI del SCADA GALBA?

De modo que se define como **objeto de estudio** el proceso de configuración en la Interfaz Hombre Máquina para los sistemas de supervisión y control.

En correspondencia con el problema planteado, se formula como **objetivo general** de este trabajo: Desarrollar un mecanismo que permita elevar el nivel de funcionalidades en el proceso de configuración del módulo HMI permitiendo construir proyectos más completos y complejos.

Definiéndose como **campo de acción** los mecanismos de configuración en la Interfaz Hombre Máquina

[1] Vistas que se desean supervisar de la industria.

para los sistemas de supervisión y control.

Para dar cumplimiento al objetivo general formulado anteriormente, se propusieron como **tareas investigativas** las siguientes:

- ✓ Establecimiento de los fundamentos teóricos-metodológicos para el desarrollo del proceso de configuración en los HMI de un sistema SCADA.
- ✓ Caracterizar el proceso de configuración en el HMI del SCADA GALBA en lo relativo a la complejidad del proceso y a la integración de los diferentes componentes.
- ✓ Establecimiento de los fundamentos necesarios para optimizar el mecanismo de configuración en el HMI del SCADA GALBA.
- ✓ Desarrollo de un mecanismo para optimizar el proceso de configuración en el HMI del SCADA GALBA.
- ✓ Validar la contribución lograda a través del mecanismo para optimizar los procesos de configuración en el HMI del SCADA GALBA.
- ✓ Valoración cualitativa de los resultados obtenidos en la validación de la optimización de los procesos de configuración en el HMI del SCADA GALBA.

Con el propósito de desarrollar las tareas planteadas se emplearon diferentes **métodos de investigación**, los cuales se explican con más detalles a continuación:

Métodos Teóricos:

- ◆ **Histórico y Lógico:** permitió explicar los antecedentes de las herramientas para el trabajo con secuencias de comandos en los sistemas SCADA y sus tendencias actuales, así como la necesidad, la importancia y los hitos que surgen del desarrollo de una herramienta como esta.
- ◆ **Análisis y síntesis:** permitió, a partir de los estudios e investigaciones hechas y de la información recogida acerca de aplicaciones similares, llegar a conclusiones sobre el problema estudiado y presentar posibles vías para solucionarlo.
- ◆ **Modelación:** permitió la elaboración de los diferentes diagramas para el entendimiento y mantenimiento de la herramienta en cuestión.

Métodos Empíricos:

- **Consultas de fuentes de información:** se utilizó este método para las consultas de fuentes bibliográficas durante la investigación.

CAPÍTULO 1
FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

Este capítulo tiene como objetivos definir y elaborar un marco teórico donde se expongan temas fundamentales que sustenten la investigación; además de hacer una valoración de las principales tecnologías y herramientas que serán empleadas para dar solución al problema planteado, definiendo las características que las hacen relevantes ante otras de su tipo. Se tratan aspectos relacionados con los sistemas SCADA, haciendo una breve historia relacionada con la evolución de estos sistemas. Además se perfilan con más detalles los conceptos que fundamentan la herramienta de gestión de script para el HMI Qt del SCADA Guardián del ALBA.

1.2 SCADA

Los sistemas SCADA son aplicaciones de software de control de producción, que se comunican con los dispositivos de campo y controlan el proceso de forma automática desde la pantalla de un ordenador u otra tecnología de comunicación.

Los sistemas SCADA son partes integrales de la mayoría de los ambientes industriales complejos o muy geográficamente dispersos ya que pueden recoger la información de una gran cantidad de fuentes muy rápidamente, y la presentan a un operador en una forma amigable[2]. Además, mejoran la eficacia del proceso de monitoreo y control, proporcionando la información oportuna para poder tomar decisiones operacionales apropiadas. (1)

Los primeros SCADA eran simplemente sistemas de telemetría que proporcionaban reportes periódicos de las condiciones de campo, vigilando las señales que representaban medidas y/o condiciones de estado en ubicaciones de campo remotas. Estos sistemas ofrecían capacidades muy simples de monitoreo y control, sin proveer funciones de aplicación alguna. La visión del operador en el proceso estaba basada en los contadores y las lámparas detrás de paneles llenos de indicadores. Mientras la tecnología se desarrollaba,

[2] Se refiere a la facilidad de uso o sencillez.

los ordenadores asumieron el papel de manejar la recolección de datos, disponiendo de comandos de control, y una nueva función: presentación de la información sobre una pantalla de CRT (del inglés Cathode Ray Tube o tubos de rayos catódicos). Los ordenadores agregaron la capacidad de programar el sistema para realizar funciones de control más complejas. (1)

Los primeros sistemas SCADA fueron altamente modificados con programas de aplicación específicos para atender a requisitos de algún proyecto particular. Como asistieron ingenieros de varias industrias al diseño de estos sistemas, su percepción de SCADA adquirió las características de su propia industria. Los proveedores de sistemas de software SCADA, deseando reutilizar su trabajo previo sobre los nuevos proyectos, perpetuaron esta imagen de industria específica por su propia visión de los ambientes de control con los cuales tenían experiencia. Solamente cuando nuevos proyectos requirieron funciones y aplicaciones adicionales, hizo que los desarrolladores de sistemas SCADA tuvieran la oportunidad de desarrollar experiencias en otras industrias. (1)

Hoy, los proveedores de SCADA están diseñando sistemas que son pensados para resolver las necesidades de muchas industrias con módulos de software industriales específicos disponibles para proporcionar las capacidades requeridas comúnmente. No es inusual encontrar software SCADA comercialmente disponible adaptado para procesamiento de papel y celulosa, industrias de aceite y gas, hidroeléctricas, gerenciamiento y provisión de agua, control de fluidos, etc. Puesto que los proveedores de SCADA aún tienen tendencia en favor de algunas industrias sobre otras, los compradores de estos sistemas a menudo dependen del proveedor para una comprensiva solución a su requisito, y generalmente procuran seleccionar un vendedor que pueda ofrecer una completa solución con un producto estándar que esté apuntado hacia las necesidades específicas del usuario final. Si selecciona a un vendedor con experiencia limitada en la industria del comprador, este debe estar preparado para asistir al esfuerzo de ingeniería necesario para desarrollar el conocimiento adicional de la industria requerido por el vendedor para poner con éxito el sistema en ejecución. (1)

Entre las funciones principales de un sistema SCADA se encuentran:

- **Adquisición de datos:** para recoger, procesar y almacenar la información recibida.
- **Supervisión:** para observar desde un monitor la evolución de las variables de control.
- **Control:** para modificar la evolución del proceso, actuando bien sobre los reguladores autónomos

básicos (consignas, alarmas, menús, etc.) o directamente sobre el proceso mediante las salidas conectadas.

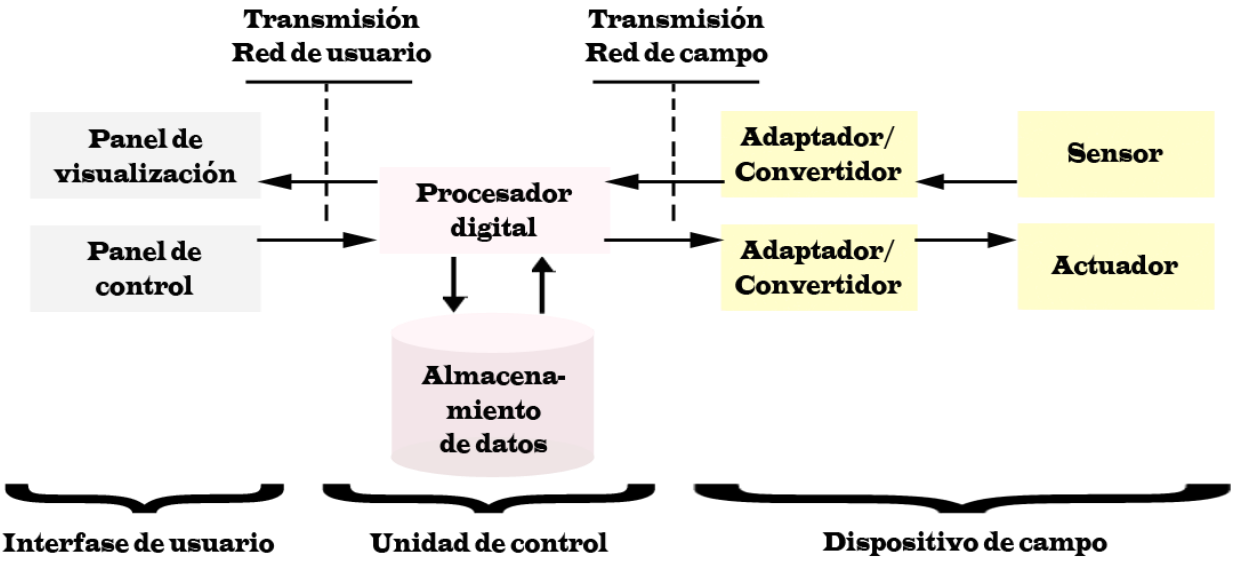

Ilustración 1 Esquema básico de un sistema de Adquisición, supervisión y control.

Además poseen funciones un poco más específicas a las cuales se hace referencia a continuación:

- **Transmisión:** De información con dispositivos de campo y otros PC.
- **Base de datos:** Gestión de datos con bajos tiempos de acceso.
- **Presentación:** Representación gráfica de los datos. Interfaz del Operador o HMI (Human Machine Interface).
- **Explotación:** De los datos adquiridos para gestión de la calidad, control estadístico, gestión de la producción y gestión administrativa y financiera.

1.3 Componentes de un sistema SCADA.

Los bloques de software o módulos que intervienen en las actividades de adquisición, supervisión y

control de los sistemas SCADA se explican a continuación:

- **Configuración:** Permite al usuario definir el entorno de trabajo de su SCADA, adaptándolo a la aplicación particular que se desea desarrollar.

- **Interfaz hombre-máquina:** Proporciona al operador las funciones de control y supervisión de procesos. El proceso se representa mediante sinópticos gráficos almacenados en el ordenador de proceso y generados desde el editor incorporado al sistema SCADA. Tradicionalmente estos sistemas consistían en paneles compuestos por indicadores y comandos, tales como luces pilotos, indicadores digitales y analógicos, registradores, pulsadores, selectores y otros que se interconectaban con la máquina o proceso. En la actualidad, dado que las máquinas y procesos en general están implementadas con controladores y otros dispositivos electrónicos que dejan disponibles puertas de comunicación, es posible contar con sistemas de HMI mucho más poderosos y eficaces, además de permitir una conexión más sencilla y económica con el proceso o máquinas.

- **Módulo de proceso:** Ejecuta las acciones de mando pre-programadas a partir de los valores actuales de variables leídas. La programación se realiza por medio de bloques de programa en lenguaje de alto nivel (como C, Basic, etc.). Presenta como principales características la adquisición de datos del nivel de recolección en tiempo real, el procesamiento de variables calculadas, la detección y el manejo de alarmas, la conversión de unidades, el control de calidad de los datos recolectados, la publicación a los clientes de los puntos y la sucesión de alarmas y eventos.

- **Gestión y archivo de datos:** Se encarga del almacenamiento y procesado ordenado de los datos, de forma que otra aplicación o dispositivo pueda tener acceso a ellos.

- **Comunicaciones o Middleware[3]:** Se encarga de la transferencia de información entre los diferentes servicios del SCADA y el resto de elementos informáticos de gestión.

[3] Software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

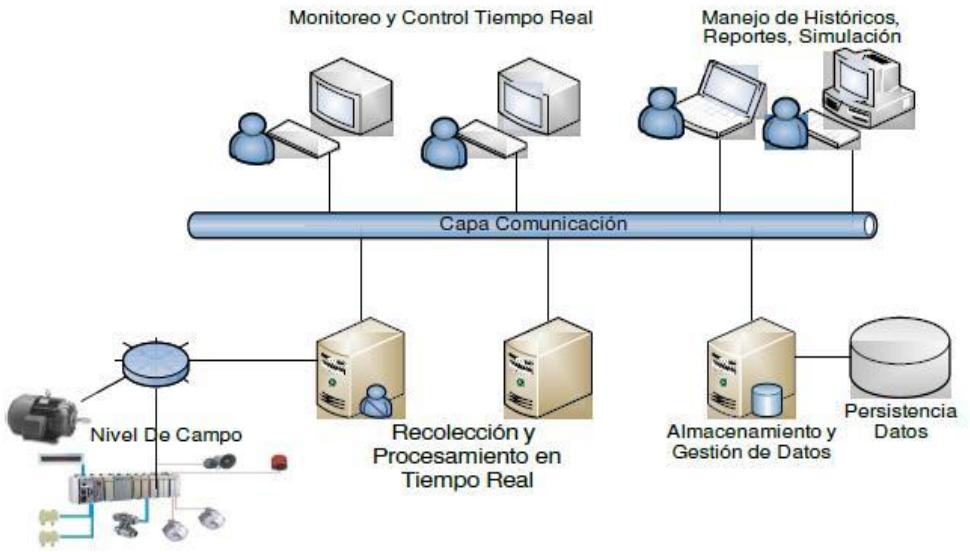

Ilustración 2 Diagrama de componentes de un sistema SCADA

1.4 Módulo Interfaz Hombre-Máquina.

Los sistemas HMI podemos pensarlos como una "ventana[4]" de un proceso. Esta ventana puede estar en dispositivos especiales como paneles de operador o una computadora. Estos sistemas, en computadoras se les conoce también como software de interfaz hombre máquina o de monitoreo y control de supervisión. Las señales de los procesos son conducidas al mismo por medio de dispositivos como tarjetas de entrada/salida en la computadora, PLC's (Controladores lógicos programables), RTU (Unidades remotas de I/O) o DRIVE's (Variadores de velocidad de motores). Todos estos dispositivos deben tener una comunicación que entienda el HMI.

- **Tipos de HMI.**
  - **Desarrollos a medida.** Se desarrollan en un entorno de programación gráfica como VC++, Visual Basic, Delphi, etc.
  - **Paquetes enlatados HMI.** Son paquetes de software que contemplan la mayoría de las funciones estándares de los sistemas SCADA. Ejemplos son FIX, WinCC, Wonderware, etc.

[4] Parte fundamental de la interfaz gráfica de la computadora, especialmente con sistemas operativos que permiten el multiprocesamiento.

➢ **Funciones de un software HMI.** (2)

- **Monitoreo:** Es la habilidad de obtener y mostrar datos en tiempo real. Estos datos se pueden mostrar como números, texto o gráficos que permitan una lectura más fácil de interpretar.
- **Supervisión:** Esta función permite junto con el monitoreo la posibilidad de ajustar las condiciones de trabajo del proceso directamente desde la computadora.
- **Alarmas:** Es la capacidad de reconocer eventos excepcionales dentro del proceso y reportar estos eventos. Las alarmas son reportadas basadas en límites de control pre-establecidos.
- **Control:** Es la capacidad de aplicar algoritmos que ajustan los valores del proceso y así mantener estos valores dentro de ciertos límites. Este va más allá del control de supervisión, removiendo la necesidad de la interacción humana. Sin embargo, la aplicación de esta función desde un software corriendo en una PC puede quedar limitada por la confiabilidad que quiera obtenerse del sistema.
- **Históricos:** Es la capacidad de muestrear y almacenar en archivos, datos del proceso a una determinada frecuencia. Este almacenamiento de datos es una poderosa herramienta para la optimización y corrección de procesos.

1.5 SCADA GALBA.

Desde el año 2006 se comienza a desarrollar en la UCI un software SCADA en convenio con la Gerencia AIT de la empresa Petróleos de Venezuela S.A (PDVSA), conocido en sus inicios como SCADA Nacional o SCADA PDVSA y a partir del 2008 en que fue presentado en la "Cumbre del ALBA" se comienza a conocer como SCADA GALBA, donde se proyectó una futura instalación del sistema en los países integrantes de esta organización. El software, que se da como solución es realizado en cooperación de distintos equipos de desarrollo de la UCI, DST-AIT PDVSA, Centro de Desarrollo de Automática Integral (CEDAI), Universidad Central "Marta Abreu" de Las Villas (UCLV), Instituto Superior Minero Metalúrgico de Moa (ISMMM), Universidad de los Andes (ULA), DBAccess[5], IntelCom[6] y la empresa de Ingeniería de Software y Calidad Aplicada (Isca).

[5] Organización latinoamericana de proyección global proveedora de servicios de Tecnología de la Información.
[6] Empresa de ingeniería prestadora de servicios en soluciones integrales de telecomunicaciones en Venezuela.

1.6 Lenguajes script integrados a sistemas SCADA.

Un script (cuya traducción literal es "guión") o archivo de órdenes o archivo de procesamiento por lotes, es un programa usualmente simple, que por lo general se almacena en un archivo de texto plano. Los script son casi siempre interpretados, pero no todo programa interpretado es considerado un script. El uso habitual de los scripts es realizar diversas tareas como combinar componentes, interactuar con el sistema operativo o con el usuario. Por este uso frecuente es que los shells[7] sean a la vez intérpretes de este tipo de programa. (3)

La aplicación de la tecnología script en un sistema SCADA provee desarrollar aplicaciones más complejas y potentes que permitan a los operadores por ejemplo: observar en la pantalla intermitencias, alarmas interactivas, movimientos de productos independientemente de su posición ya sea horizontal o vertical, llenado y vaciado de productos, flujos, secuencias, funciones matemáticas, es decir, todo lo que el diseñador pueda requerir.

1.6.1 Visual Basic for Applications.

Visual Basic for Applications (VBA) es el lenguaje de programación incorporado en Microsoft Office. Es un lenguaje muy extendido y se ha convertido en un estándar de facto, que permite la integración de aplicaciones de terceros y la comunicación directa con cualquier aplicación de MSOffice y de cualquier aplicación compatible con VBA. Este lenguaje se ha extendido y ha sido aceptado por la mayoría de los usuarios. Simatic WinCC de SIEMENS es un ejemplo de ello al integrar VBA, en el WinCC Graphics Designer. Por otro lado GENESIS32 de Iconics no solo integra esta característica a su editor gráfico, sino que ofrece el módulo ScriptWorX32 que es una poderosa herramienta para crear y ejecutar programas en VBA, que pueden correr simultáneamente basados en intervalos de tiempos o de eventos. El uso de un lenguaje común también facilita la integración de objetos suministrados por terceros. Además permite interactuar directamente con las aplicaciones de Office (Access, Excel y Word) y con otros productos compatibles. Por ejemplo existen bibliotecas de funciones para obtener y cambiar datos que se

[7] Intérprete de órdenes, son aplicaciones capaces de interpretar las órdenes del usuario a través de comandos escritos, como por ejemplo el sistema MS-DOS o los terminales de consola de los sistemas operativos Linux. Estas aplicaciones permiten al usuario interactuar con el ordenador, normalmente a través de una sencilla interfaz de texto plano y suponen la forma más básica de interacción de un usuario con su ordenador, escribiendo las órdenes en este Shell a través de comandos y recogiendo las respuestas de la máquina.

encuentran en los servidores OPC. (4)

Debido a la integración de VBA en diferentes aplicaciones, se logra que aumente gradualmente la conexión entre SCADA y programas de gestión. Una de las ventajas del uso de VB sobre C++ o algún otro idioma es la disponibilidad de programadores de VB, otra es la velocidad con la que se pueden desarrollar aplicaciones. Sin embargo también tiene limitantes, una de ellas es la latencia, es decir, tiende a ser un poco más lenta, en algunos casos de C++ y otros idiomas.

```
Public Sub Global0()
    Const OPCTag_Global0 = "Moeller.S40-OPC-DataAccess.2\ALAMAL1"

    'crear objeto de tipo datapoint si no existe
    Call gSwxStorage.CreateDataPoint(OPCTag_Global0,"Undefined"..
                                ,,,"Undefined")

    'crear objeto para acceder al dato en el servidor OPC
    Dim OPCTag As Swx.IDataPoint
    Set OpcTag = gSwxStorage.AccessDataPoint(OPCTag_Global0)
    If OPCTag Is Nothing Then
        gSwxGlobal.PrintToConsole "No se puede crear objeto DataPoint"
        Exit Sub
    End If

    Dim Value, TimeStamp, TimeStamp_ms As Variant
    Dim Quality As Swx.tagDATAPOINT_QUALITY
    Value = OPCTag.Read(Quality, TimeStamp, TimeStamp_ms)
    Select  Case Quality
    Case tagDATAPOINT_QUALITY.QUALITY_BAD
        Range("B" + CStr(9 + i -1)).value = "Fallo"

    Case tagDATAPOINT_QUALITY.QUALITY_GOOD
        Range("B" + CStr(9 + i -1)).value = value

    Case tagDATAPOINT_QUALITY.QUALITY_UNCERTAIN
        Range("B" + CStr(9 + i -1)).value = "Impreciso"

    End Select
    Exit Sub
End Sub
```

Ilustración 3 Fragmento de código en VBA para mostrar el valor de una variable de un servidor OPC en una hoja Excel.

1.6.2 HMI Scripting Language.

El HMI Scripting Language utiliza la misma sintaxis y semántica que el Microsoft Visual Basic. Permite la interacción con otras aplicaciones de Windows, además de realizar múltiples acciones como la conversión de unidades, formato e impresión de reportes, la gestión de cuadros de diálogos y entradas de usuario, así como la alternación entre las pantallas del operador. (5)

1.6.3 C++ Script.

Es una biblioteca de C++ que proporciona tipos de secuencias de comando, dinámicos y programación dinámica en C++. El paradigma dinámico permite a C++ ser utilizado de una manera más sencilla, sin requerir el conocimiento de la gestión de memoria, clases, plantillas, los contenedores y los iteradores. Muchos problemas pueden ser expresados de una manera más sencilla utilizando los tipos dinámicos. Como lenguajes de secuencia de comandos, programas en C++ Script se compilan utilizando el estándar de C++, puede incrustar en el estándar código de C para lograr un alto rendimiento. (6)

1.6.4 Qt Script.

Qt Script se basa en el lenguaje de scripting ECMAScript, como se define en el estándar ECMA-262. ECMAScript es el nombre oficial del lenguaje estandarizado por ECMA Internacional. Constituye la base de JavaScript (Mozilla), JScript (Microsoft) y ActionScript (Adobe). Aunque la sintaxis del lenguaje es superficialmente similar a C++ y Java, los conceptos subyacentes son radicalmente diferentes y que lo distinguen de la mayoría de otros lenguajes de programación orientados a objetos. Las estructuras para el control básico en ECMAScript son las mismas que en C++ y Java; las sentencias if, los bucles for y while. Además, proporciona más o menos la misma asignación, relacionales y operadores aritméticos. Las cadenas definidas por el lenguaje soportan concatenación con + y añadir valores con +=. (7)

El marco de trabajo gráfico Qt a partir de su versión 4.3 proporciona soporte para aplicaciones de secuencias de comandos utilizando ECMAScript. Este lenguaje está lejanamente relacionado con Qt Script for Applications (QSA). Sus extensiones son mucho más fáciles que QSA y es más flexible y eficiente. Proporciona un entorno de scripting embebido. Qt Script es completamente orientado a objetos, con un modelo de ellos muy similar al de Qt. Es un lenguaje moderno que cuenta con características tales como: alto nivel, tipos de datos y control de excepciones, y proporciona completamente la API[8] de Qt. Proporciona funcionalidades más ricas que las requeridas por ECMAScript, por ejemplo la clase String de Qt Script, proporciona todas las funcionalidades de QString además de las definidas en ECMAScript. (8)

[8] Interfaz de programación de aplicaciones.

1.7 Herramientas de scripting desarrolladas para sistemas SCADA.

Al aplicar un sistema SCADA, principalmente en la industria, tendremos la necesidad de crear HMI que nos permita observar de forma animada y amigable el funcionamiento de los equipos y procesos, para poder personalizar estas funciones, necesitamos programarlas a través de una lógica operacional o comandos. Para realizar estas nuevas tareas se integran a los sistemas SCADA herramientas de gestión de script realmente potentes, que facilitan el trabajo de los operadores.

1.7.1 HMI Script Editor.

QuickScript es el lenguaje scripting para HMI de InTouch. Puede ser utilizado para construir aplicaciones más robustas. Posee siete tipos de script definidos y muchas funciones integradas de secuencias de comandos disponibles. Los siete tipos de script se definen según las causas por las que se van a ejecutar. Por ejemplo, los script de aplicación se ejecutan cuando inicia la aplicación, detiene o continúa en funcionamiento. Las funciones de script incorporadas incluyen funciones matemáticas, funciones trigonométricas, funciones de cadenas, y otras. El uso de estas funciones le permite al usuario ahorrar tiempo en el desarrollo de la aplicación. Los scripts de InTouch pueden incluir vinculación y empotramientos de objetos (OLE) y controles ActiveX[9]. Puede utilizar sentencias condicionales, bucles y variables locales en el lenguaje de scripting para crear efectos complejos en su aplicación. (9)

Los tipos de script que define InTouch para su aplicación son:

- **Script de Aplicación:** Se ejecuta de forma contínua mientras que WindowViewer se está ejecutando, o una vez que WindowViewer se inicia o se detiene.
- **Script de Ventana:** Se ejecuta periódicamente cuando en InTouch una ventana está abierta o una vez que una ventana se abre o se cierra.
- **Script de Tecla:** Se ejecuta una vez o periódicamente cuando una tecla o una combinación de ellas es presionada o liberada.
- **Script de Condición:** Se ejecuta una vez o periódicamente cuando cierta condición se cumple o no.
- **Script de Cambio de Datos:** Se ejecuta una vez cuando un valor de una etiqueta o de una

[9] Entorno para definir componentes de software reusables de forma independiente del lenguaje de programación.

expresión varía.

- **Script de Acción:** Se ejecuta una vez o periódicamente cuando un operador hace clic en un objeto gráfico del HMI.
- **Script de evento de ActiveX:** Se ejecuta una vez que un evento de ActiveX ocurre, como clic en el control de ActiveX.

Algunas funciones avanzadas de scripting permiten alcanzar sofisticadas funciones más allá de las del HMI básico de InTouch. Los objetos OLE y los controles de ActiveX le permiten acceder a las funciones nativas del sistema informático e interactuar con otros programas como el Módulo de Ingeniería de Fabricación.

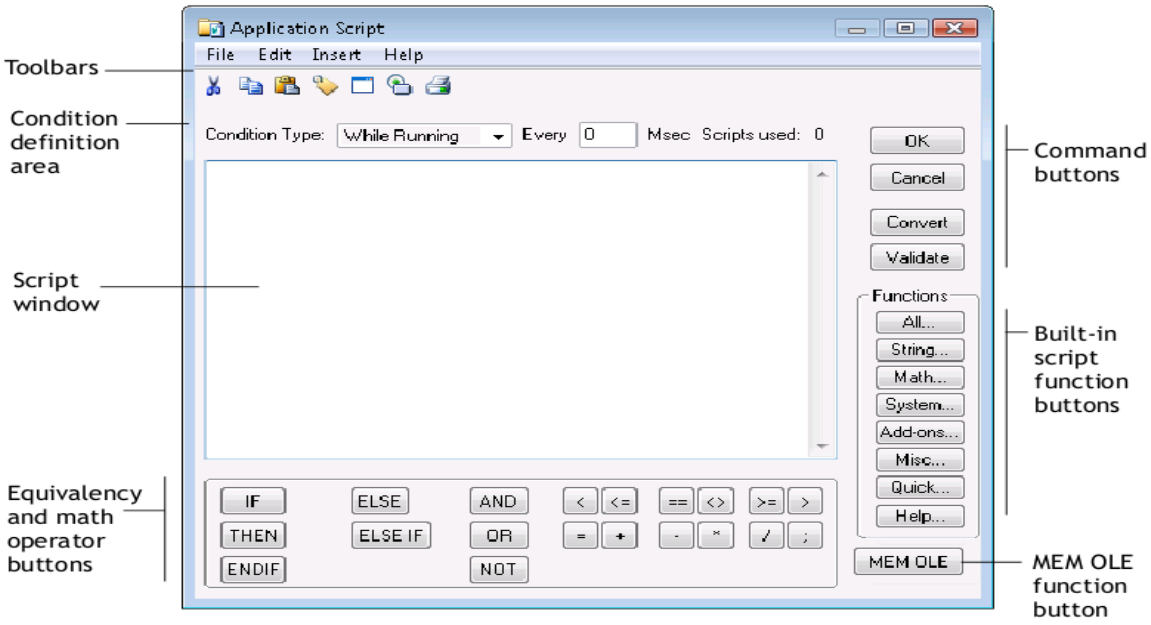

Ilustración 4 Editor script del HMI de InTouch.

1.7.2 CX-Supervisor Script Editor.

Los script, controlan las acciones de los objetos, páginas o proyectos; pueden ser creados o actualizados utilizando el cuadro de diálogo del Editor de Script de CX-Supervisor. El editor de script permite crear o modificar las operaciones que se llevan a cabo en las animaciones mediante una secuencia de comandos. Está codificado por colores para ayudar a mostrar la sintaxis correcta con las palabras claves y los

diferentes tipos de objetos se muestran en colores diferentes. Al crear una secuencia de comandos debe elegir una acción, función, etc. Además de tener un menú contextual del cual puede requerir información adicional. (10)

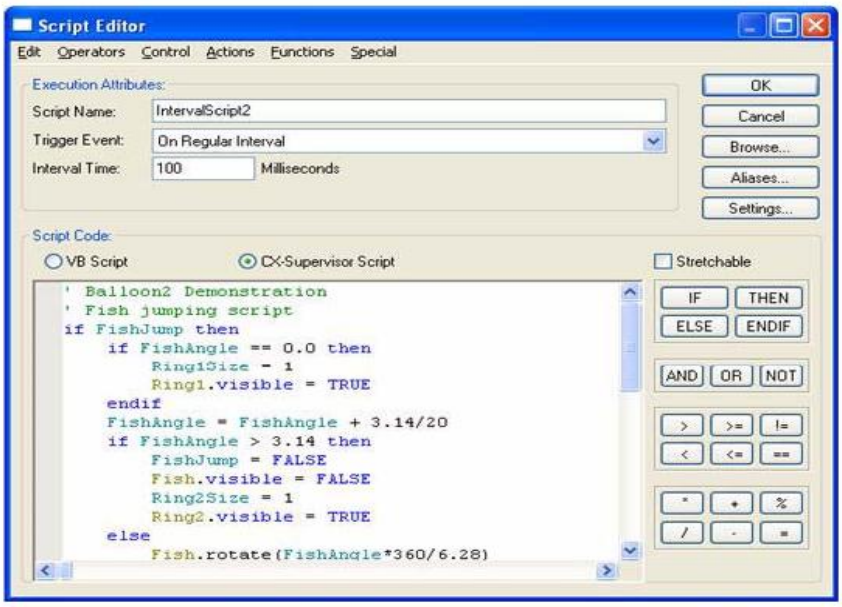

Ilustración 5 Script Editor del sistema CX-Supervisor.

1.7.3 Global Script.

El Global Script es un compilador de C incorporado en WinCC que nos permite realizar compilaciones de código objeto (con un formato un tanto especial, ya que no lo deja en obj) sin necesidad de salir del propio entorno. Este código generado puede ser añadido directamente al propio runtime[10] y generarse cada cierto tiempo, o posteriormente asociarse a un evento de un objeto. Si desde el editor del GraphicDesigner seleccionamos realizar una acción en C, automáticamente se nos activa el Global Script para permitirnos editar dicha acción en el compilador de C. (11)

Tipos de funciones en Global Script:

Dentro del compilador de Global Script podemos observar que existen cuatro tipos de clases o categorías

[10] Sub-modulo de ejecución en tiempo real de la aplicación.

de funciones:

- **Project functions:** Son aquellas funciones que deben ser llamadas desde otras partes del programa para devolver valores después de realizar alguna operación en C.
- **Standard functions:** son las funciones estándar del propio WinCC, que no residen en nuestro proyecto, sino en el subdirectorio aplib de WinCC. Estas funciones estándares se pueden modificar, pero dichas modificaciones son permanentes hasta que no se reinstale el WinCC.
- **Internal functions:** Funciones que realizan acciones predeterminadas, como son proporcionar valor de una variable, o asignar el valor a una variable.
- **Actions:** Una acción es una subrutina que no se ejecuta cuando es llamada por un evento, sino cuando se desencadena una acción, ya sea por el tiempo o por un cambio de valor de alguna variable.

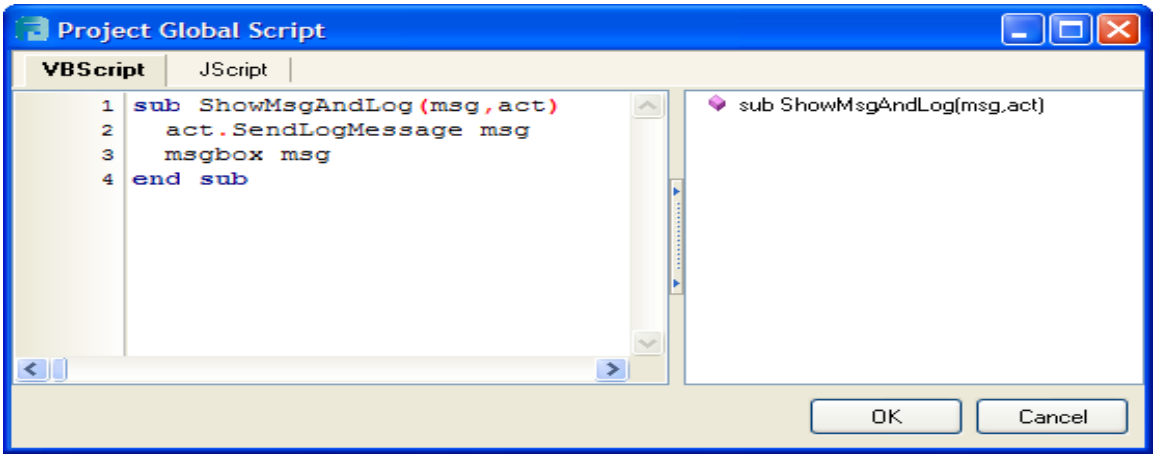

Ilustración 6 Global Script de WinCC.

1.7.4 Script Explorer.

El Script Explorer es un la herramienta para la gestión de scripts incorporado al SCADA Movicon. Esta herramienta provee un conjunto de funciones para la personalización de animaciones, por ejemplo, además de la animación habitual Movicon provee un conjunto de comandos preestablecidos que pueden asociarse al objeto Tendencia que pueden ser utilizados para personalizar el comportamiento de dicho objeto. Estos comandos están disponibles en la ventana de miembros de automatización ActiveX en las opciones de “Tipo de Datos” en el marco de conjunto de comandos de VBA, llamado “TrendCmdTarget”, al

que se accede al editar el código del objeto a través del Script Explorer. Además permite insertar eventos personalizados, asociados a las variaciones de las variables de la base de datos en tiempo real de Movicon, incluyendo las estándar que pone a disposición el SCADA (Click, DoubleClick, KeyDown, etc.) y símbolos internos del código script (Dibujos o controles). Un evento puede ser añadido para ser ejecutado cada vez que cambia el estado seleccionado de una variable. Por lo tanto, el programador desarrolla el código interno a insertar en el evento según considere necesario. El evento insertado entonces estará activo y se procesa cuando el símbolo está activo, debido a que está cargado en la RAM. (12)

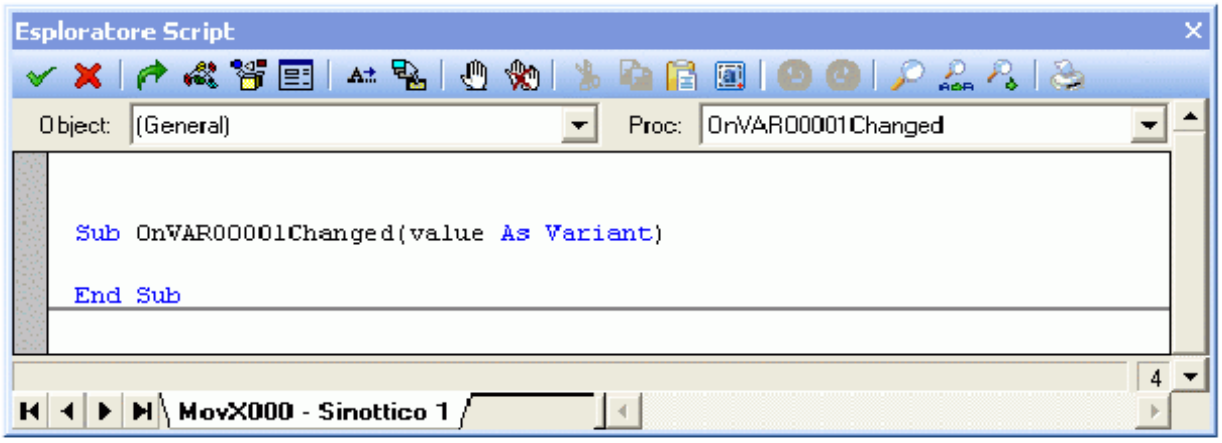

Ilustración 7 Script Explorer de Movicon.

1.8 Selección de tecnologías y metodología de desarrollo.

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

1.8.1 Sistema Operativo: GNU/Linux

La distribución seleccionada es Debian, específicamente la versión 6.0 Squeeze. Se selecciona este sistema operativo porque provee alta calidad tecnológica y gran seguridad, con menos errores que los demás sistemas comerciales y estable para el usuario promedio. Se compone de múltiples paquetes de software normalmente distribuidos bajo una licencia libre o de código abierto. Además se encuentra muy

relacionado con el SCADA GALBA, dado que este producto está siendo desarrollado sobre esta plataforma dependiendo de algunas de sus bibliotecas.

1.8.2 Ambiente de desarrollo.

El ambiente de desarrollo seleccionado es el Eclipse Juno. El mismo es un entorno de desarrollo integrado (IDE) que abarca todo el ciclo de desarrollo de software. Es una gran estructura conformada por un núcleo y varias extensiones que componen la aplicación final. Inicialmente este software se concibió para los programadores que utilizaban el lenguaje Java, pero en la actualidad existe soporte para varios lenguajes de programación como C/C++, Python, PHP, entre otros. Posee además, herramientas para el modelado de software, sistemas de gestión de bases de datos (DBMS), para la gestión de la configuración y el control de versiones, Subversion y ficheros CSV (del inglés comma-separated values). Además permite una estructura de carpetas para ordenar física y lógicamente el código fuente, la gestión de errores, así como la gestión de plantillas a partir de un estándar de código. (13)

1.8.3 Lenguaje de programación para la implementación de la solución.

Se selecciona el lenguaje de programación C++ para la implementación del mecanismo por ser el lenguaje utilizado en el desarrollo del sistema SCADA GALBA y así lograr una mejor integración de la solución con el producto final.

C++ está considerado por muchos como el lenguaje de programación más potente, dado que permite el trabajo tanto a alto nivel como a bajo nivel, logrando gran eficiencia en los tiempos de ejecución y bajo consumo de memoria en los programas desarrollados, aspectos que muchas aplicaciones requieren, siendo una opción factible como lenguaje a utilizar en sistemas que necesitan un alto rendimiento. Entre las principales características que brinda C++ se puede mencionar la programación orientada a objetos. La posibilidad de orientar la programación a objetos le permite al programador, diseñar aplicaciones desde un punto de vista más cercano a la vida real. Además de permitir reutilizar el código de una manera más lógica y productiva. (14)

1.8.4 Marco de trabajo gráfico Qt.

Qt es un marco de trabajo multiplataforma para el desarrollo de interfaces gráficas, también para el

desarrollo de aplicaciones sin interfaz gráfica como servidores; escritas en código C++. Qt, además, es completamente orientado a objetos. Este se basa en los conceptos de componentes gráficos que proporcionan las funcionalidades estándar de interfaz gráfica de usuario, introduce una innovadora alternativa para la comunicación entre objetos, conocidos como señales y ranuras (signals/slots). También propone el patrón de diseño modelo/vista (Model/View) para la representación gráfica de los datos con la separación de las funcionalidades introducidas por esta arquitectura, ofreciendo una mayor flexibilidad a los desarrolladores para personalizar la presentación de elementos y proporciona una interfaz de modelo estándar que permite una amplia gama de datos. (15)

Se puede destacar que comenzó como un marco para el desarrollo de interfaces gráficas, pero ya el API cuenta con tecnologías que facilitan el trabajo de los desarrolladores tales como:

- Soporte para hilos.
- Sistema de pintado.
- Integración de documentación en aplicaciones.
- Modelos dinámicos de objetos.
- Serialización y trabajo con DOM.
- Trabajo con extensiones.
- Contenedores genéricos.
- Estilos.
- Eventos y filtrado de eventos.

Otras características a tener presentes son: la disponibilidad de código fuente, la excelente documentación organizada que provee en un asistente (QtAssistant) y un editor para el diseño de formularios visualmente (QtDesigner). (15)

1.8.5 Metodologías de desarrollo de software.

El papel preponderante de las metodologías es sin duda esencial en un proyecto y en el paso inicial, que debe encajar en el equipo, guiar y organizar actividades que conlleven a las metas trazadas en el grupo. A continuación se detallan los dos grandes enfoques, las metodologías tradicionales y las metodologías ágiles.

- **Metodologías Tradicionales.**

Las metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada. Además, dichas metodologías no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno donde los requisitos no pueden predecirse o bien pueden variar.

Entre las metodologías tradicionales más utilizadas podemos referirnos a:

- RUP (Rational Unified Procces).
- MSF (Microsoft Solution Framework).
- Win-Win Espiral Model.
- Iconix.

- **Metodologías Ágiles.**

En una reunión celebrada en febrero del 2001 en Utha-EEUU, nace el término “ágil” aplicado al desarrollo de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar aplicaciones rápidamente y responder a cambios que pudieran surgir a lo largo del proyecto, pretendiendo ofrecer una alternativa a los procesos de desarrollo de software tradicionales. (16)

Entre las metodologías ágiles más destacadas hasta el momento se pueden nombrar:

- XP (Extreme Programming).
- Scrum.
- Crystal Clear.
- DSDM (Dynamic Systems Development Method).
- FDD (Feature Driven Development).

- ASD (Adaptative Software Development).
- XBreed.
- Extreme Modeling.

➢ **Revisión de metodologías.**

Aunque los creadores e impulsores de las metodologías ágiles más populares han subscrito el manifiesto ágil y coinciden con los principios enunciados anteriormente, cada metodología tiene características propias y hace hincapié en algunos aspectos más específicos. A continuación se resumen las principales metodologías ágiles:

- **SCRUM:** Desarrollada por Kent Schwaber, Jeff Sutherland y Mike Beedle. Define un marco para la gestión de proyectos, que se ha utilizado con éxito durante los últimos 10 años. Está especialmente indicada para proyectos con un rápido cambio de requisitos. Sus principales características se pueden resumir en dos; el desarrollo de software se realiza mediante iteraciones denominadas sprints y con una duración de 30 días, la segunda característica importante son las reuniones a lo largo del proyecto. (16)

- **Crystal Methodologies:** Se trata de un conjunto de metodologías para el desarrollo de software caracterizadas por estar centradas en las personas que componen el equipo (de ellas depende el éxito del proyecto) y la reducción al máximo del número de artefactos producidos. Han sido desarrolladas por Alistair Cockburn. El desarrollo de software se considera un juego cooperativo de invención y comunicación, limitado por los recursos a utilizar. El equipo de desarrollo es un factor clave, por lo que se deben invertir esfuerzos en mejorar sus habilidades y destrezas, así como tener políticas de trabajo en equipos definidas. Estas políticas dependerán del tamaño del equipo, estableciéndose una clasificación por colores, por ejemplo Crystal Clear (3 a 8 miembros) y Crystal Orange (25 a 50 miembros). (16)

- **Dynamic Systems Development Method:** Define el marco para desarrollar un proceso de producción de software. Creada en 1994 con el objetivo de elaborar una metodología RAD unificada. Sus principales características son: es un proceso iterativo e incremental y el equipo de desarrollo y el usuario trabajan juntos. Propone cinco fases: estudio viabilidad, estudio del negocio,

modelado funcional, diseño y construcción, y finalmente implementación. Las tres últimas son iterativas, además de existir realimentación a todas las fases. (16)

- **Feature-Driven Development:** Define un proceso iterativo que consta de cinco pasos. Las iteraciones son cortas (hasta 2 semanas). Se centra en las fases de diseño e implementación del sistema partiendo de una lista de características que debe reunir el software. Sus impulsores son Jeff De Luca y Peter Coad. (16)

- **Extreme Programming:** La Programación Extrema es una metodología ligera de desarrollo de software que se basa en la simplicidad, la comunicación y la retroalimentación o reutilización del código desarrollado. Fue creada a mediados de la década de los 80 por Kent Beck. Según su autor es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo del software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se define como especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. (17)

  Los objetivos de XP son muy simples: el primero la satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita, por tanto se debe responder rápidamente las necesidades del cliente, incluso cuando los cambios sean al final del ciclo de programación. El segundo objetivo es potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y los desarrolladores, son parte del equipo y están involucrados en el desarrollo de software. (17)

  XP define cuatro variables para proyectos de software: coste, tiempo, calidad y ámbito. Además de estas cuatro variables, Beck propone que solo tres de ellas pueden ser establecidas por las fuerzas externas (jefes de proyecto y clientes), mientras que el valor de la cuarta variable debe ser establecido por los programadores en función de las otras tres. Además define cuatro actividades básicas para el desarrollo de software, ellas son: codificar, hacer pruebas, escuchar y diseñar. El ciclo de vida ideal de XP consiste en 6 fases: Exploración, Planificación de entrega, Iteraciones, Producción, Mantenimiento y Muerte del proyecto. Algunos de los beneficios de utilizar las prácticas de la programación extrema son: programación en pares, refactorización, integración

continua, pruebas de aceptación, unidad de pruebas y otras que favorecen el aprendizaje de la programación e incitan a que se realicen investigaciones que apoyen la integración de estas prácticas. (17)

La tabla siguiente compara las distintas aproximaciones ágiles en base a tres parámetros: vista del sistema como algo cambiante, tener en cuenta la colaboración entre los miembros del equipo y características más específicas de la propia metodología como simplicidad, excelencia técnica, resultados, adaptabilidad, etc. También incorpora como referencia no ágil el Capability Madurity Model (CMM).

	CMM	**ASD**	**Crystal**	**DSDM**	**FDD**	**LD**	**Scrum**	**XP**
Sistema como algo cambiante.	1	5	4	3	3	4	5	5
Colaboración.	2	5	5	4	4	4	5	5
Características Metodología (CM)								
Resultados.	2	5	5	4	4	4	5	5
Simplicidad.	1	4	4	3	5	3	5	5
Adaptabilidad.	2	5	5	3	3	4	4	3
Excelencia técnica.	4	3	3	4	4	4	3	4
Prácticas de colaboración.	2	5	5	4	3	3	4	5
Media Total	1.7	4.8	4.5	3.6	3.6	3.9	4.7	4.8

Tabla 1 Ranking de “agilidad” (Los valores más altos representan una mayor agilidad)

Como se observa en la tabla, todas las metodologías ágiles tienen una significativa diferencia del índice de agilidad respecto a CMM y entre ellas destacan ASD, Scrum y XP como las más ágiles.

La metodología de desarrollo XP, por sus características, es la que más se ajusta para modelar la herramienta que se propone como solución, ya que por su definición es una metodología ágil, utilizada por pequeños grupos de desarrollo, para proyectos de corto período de tiempo.

1.8.6 Lenguaje de modelado.

El Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software. (18)

Está compuesto por varios elementos gráficos que se combinan mediante reglas para conformar los diferentes diagramas. Permite además la modelación de sistemas con tecnología orientada a objetos. La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Es importante destacar que el modelo UML describe lo que supuestamente hará el sistema, pero no dice cómo implementar dicho sistema. (18)

Por todo lo anteriormente mencionado y teniendo en cuenta además, que para el desarrollo de la aplicación propuesta en la solución se seguirá el paradigma orientado a objeto; se ha decidido que el lenguaje de modelado a utilizar sea el UML.

1.8.7 Herramienta para el modelado.

En la actualidad las tecnologías ayudan considerablemente a solventar muchos detalles que en años previos retrasaban el desarrollo. Entre las tecnologías mencionadas anteriormente se encuentra las herramientas de modelado, estas constituyen el medio donde se modela el sistema que se desea, rigiéndose por una metodología y utilizando algún lenguaje de modelado.

Una herramienta CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) es un conjunto de métodos, utilidades y técnicas que proporcionan la automatización del ciclo de vida del desarrollo de sistemas de información, completamente o en algunas de sus fases. Estas herramientas engloban todos los pasos del desarrollo de software, y también aquellas actividades generales que se aplican a lo largo del proceso.

Se selecciona para el modelado del sistema el Visual Paradigm 8.0, dado que es una herramienta CASE para UML, de fácil uso y completa, con soporte multiplataforma, y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Es un producto de calidad, soporta aplicaciones Web, se

puede encontrar en varios idiomas, apoya un gran número de idiomas en la generación de código y la ingeniería inversa en Java, C++, PHP, entre otros, así como exportación como HTML, es fácil de instalar y de actualizar y posee compatibilidad entre ediciones. Esta herramienta que está diseñada para usuarios interesados en sistemas de software de gran escala con el uso de la aproximación orientado a objeto, incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros de equipo. (19)

Además se decide utilizar Visual Paradigm 8.0 debido a que ofrece:

- ✓ Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- ✓ Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- ✓ Capacidades de ingeniería directa e inversa.
- ✓ Disponibilidad de integrarse en los principales entornos de desarrollo.
- ✓ Disponibilidad de múltiples versiones para cada necesidad.
- ✓ Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.

1.8.8 Lenguaje de programación script para la creación de secuencias de comandos.

En informática, un lenguaje interpretado es un lenguaje de programación que está diseñado para ser ejecutado por un intérprete. A ciertos lenguajes interpretados también se les conoce como lenguaje script. Los lenguajes script proveen a las aplicaciones cierta flexibilidad adicional sobre otros lenguajes de programación, así como una gestión de memoria automática.

Del estudio realizado anteriormente y después de analizados los lenguajes script más utilizados en los sistemas SCADA, se profundiza en el estudio de tres de ellos por las características presentadas, los mismos son Visual Basic for Applications, C++ Script y Qt Script. De los lenguajes seleccionados se define la utilización de Qt Script como lenguaje base para la generación de código script por ser un lenguaje multiplataforma, además permite la programación orientada a objetos con un potente tratamiento de eventos y posibilitar una integración al SCADA GALBA más sencilla permitiendo un ahorro en tiempo y esfuerzo.

1.9 Consideraciones parciales

Este capítulo permitió cumplir los objetivos definidos para el mismo, consolidando conceptos referentes a los sistemas SCADA y tecnologías scripting aplicadas a estos sistemas. Además se definieron y seleccionaron las tecnologías necesarias para llevar a cabo el cabal cumplimiento del objetivo planteado en el trabajo. Basado en el análisis realizado de las informaciones que describen las posibles tecnologías y metodologías a utilizar en la aplicación propuesta en la solución, se selecciona como sistema operativo Debian Squeeze por su singularidad de pertenecer a la categoría de software libre; como biblioteca gráfica Qt; Qt Script como lenguaje script para la elaboración de secuencias de comandos; el lenguaje de programación C++ para implementar la solución y Eclipse como entorno de desarrollo integrado. UML como lenguaje de modelado y Visual Paradigm como herramienta CASE. La metodología que guiará la evolución del desarrollo será XP por la flexibilidad y documentación de la misma.

CAPÍTULO 2

ANÁLISIS Y DISEÑO DEL MECANISMO DE SCRIPTING

2.1 Introducción

El presente capítulo tiene como objetivo reflejar las actividades realizadas en los procesos de análisis y diseño del mecanismo de scripting. En el mismo se exponen los artefactos más importantes que describen el flujo normal de eventos que ocurren en el sistema tales como especificaciones de requisitos que rigieron el desarrollo de la solución, historias de usuarios, diagramas y las principales clases que componen el mecanismo, detallando la información del análisis y del diseño de la solución en cuestión.

2.2 Análisis de la solución propuesta.

Partiendo de la situación problémica y el estudio realizado acerca de la tecnología script a utilizar en el mecanismo propuesto como solución, se definieron como principales requisitos funcionales (RF) del mecanismo de integración de scripting al HMI los listados a continuación. Teniendo presente que un requisito funcional define el comportamiento interno del software, es decir, funcionalidades específicas de la aplicación; son complementados por los requisitos no funcionales, que se enfocan en el cambio en el diseño o en la implementación.

- **Requisitos funcionales:**

  - RF1 Gestionar un script.
    - RF1.1 Adicionar script.
    - RF1.2 Modificar script.
    - RF1.3 Eliminar script.
  - RF2 Asociar script a los recursos y controles.
    - RF2.1 Asociar script a los recursos.
    - RF2.2 Asociar script a los controles.
  - RF3 Personalización de propiedades y animaciones.
    - RF3.1 Personalización de propiedades.

  - RF3.2 Personalización de animaciones.
- RF4 Asociar variables a los componentes gráficos.
- RF5 Crear objetos gráficos mediante scripting y adicionarlos al despliegue.
  - RF5.1 Crear objetos gráficos mediante scripting.
  - RF5.2 Agregar objetos gráficos al despliegue mediante scripting.
- RF6 Salvar e importar scripts.
  - RF6.1 Salvar scripts.
  - RF6.2 Importar scripts.
- RF7 Depuración[11] del código script.

Requisitos no funcionales:

- Usabilidad
  - RNF1 La aplicación debe ser amigable y fácil de utilizar.
- Software
  - RNF2 El mecanismo debe ser implementado en el sistema operativo Debian.
  - RNF3 El mecanismo debe estar implementado de forma óptima e incorporado al SCADA GALBA.
- Hardware
  - RNF4 El mecanismo debe instalarse junto con el SCADA en un ordenador con al menos 1 GB de memoria RAM.
- Diseño e implementación.
  - RNF5 Permitir el completamiento de código en el editor.
  - RNF6 Resaltar palabras reservadas del lenguaje.

2.2.1 Roles de la metodología XP.

Se definió como persona relacionada con el sistema, a aquella que interactúa de una forma u otra con el mecanismo propuesto, dígase vinculado al proceso de desarrollo, así como a la persona que interactúa con la herramienta de gestión de script. (16)

[11] Es el proceso de identificar y corregir errores en la programación.

Roles	Descripción
Programador	Es la persona encargada de realizar la implementación de la herramienta de programación visual y desarrollar todas las especificaciones que requiera el cliente.
Cliente	Mantenedor: Es la persona encargada de gestionar e implementar los script y mejorar las funcionalidades del sistema.

Tabla 2 Roles de la metodología XP.

2.3 Fase de exploración.

La metodología XP, comienza en su ciclo de vida con la fase de exploración, proponiendo definir durante esta etapa el alcance general del proyecto; además, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo se realizó una familiarización con las herramientas, tecnologías y prácticas que se emplearon en el proyecto. Se probó la tecnología y se exploró las posibilidades de la arquitectura del sistema construyendo un prototipo. Las estimaciones realizadas en esta fase son primarias, ya que estuvieron basadas en datos de alto nivel y podrán variar cuando se analicen con mayor detalle en cada iteración. (16)

2.3.1 Historias de usuarios.

Las historias de usuario son la técnica utilizada en XP para especificar los requisitos del sistema a desarrollar. El cliente describió y priorizó sus necesidades mediante estas descripciones cortas y escritas sin terminología técnica. Se realizaron una por cada funcionalidad del sistema, se emplearon para hacer estimaciones de tiempo y para el plan de lanzamientos. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores pudieran implementarla en unas semanas. Las historias de usuarios se programaron en un tiempo entre una y tres semanas. (16)

Como resultado del trabajo realizado durante la fase de exploración se identificaron las siguientes historias de usuario:

Historia de usuario	
Número: 1	**Nombre:** Gestionar script.
Usuario: Desarrollador	
Iteración asignada: 1	**Prioridad del Negocio:** Alta
Puntos estimados: 2	**Puntos reales:** 2
Descripción: Se debe dotar al editor de un mecanismo para adicionar, editar y eliminar scripts en el proyecto. Estos podrán involucrar recursos, objetos y controles que disponga la aplicación.	

Tabla 3 HU Gestionar script.

Historia de usuario	
Número: 2	**Nombre:** Asociar script a los recursos y controles.
Usuario: Desarrollador	
Iteración asignada: 1	**Prioridad del Negocio:** Alta
Puntos estimados: 1	**Puntos reales:** 1
Descripción: Permitir al editor, asociarle a los recursos y controles disponibles en la aplicación, secuencias de comandos.	

Tabla 4 HU Asociar script a los recursos y controles.

Historia de usuario	
Número: 3	**Nombre:** Personalización de propiedades y animaciones.
Usuario: Desarrollador	
Iteración asignada: 2	**Prioridad del Negocio:** Alta
Puntos estimados: 1	**Puntos reales:** 1
Descripción: Permitir al editor utilizar códigos script para modificar las propiedades de los objetos gráficos; posibilitando asociar, cambiar o definir comportamientos para dicho objeto.	

Tabla 5 HU Personalización de propiedades y animaciones.

Historia de usuario	
Número: 4	**Nombre:** Asociar variables a los componentes gráficos.
Usuario: Desarrollador	
Iteración asignada: 2	**Prioridad del Negocio:** Alta
Puntos estimados: 1	**Puntos reales:** 1
Descripción: Permitir al editor, asociarle a los componentes gráficos existentes en un despliegue, los valores de los puntos definidos por el módulo de Adquisición da datos.	

Tabla 6 HU Asociar variables a los componentes gráficos.

Historia de usuario	
Número: 5	**Nombre:** Depuración del código script.
Usuario: Desarrollador	
Iteración asignada: 2	**Prioridad del Negocio:** Media
Puntos estimados: 1	**Puntos reales:** 1
Descripción: Permitir al editor de script, ejecutar el código script en modo de depuración para identificar posibles errores sintácticos y semánticos.	

Tabla 7 HU Depuración del código script.

Historia de usuario	
Número: 6	**Nombre:** Crear objetos gráficos mediante scripting.
Usuario: Desarrollador	
Iteración asignada: 3	**Prioridad del Negocio:** Media
Puntos estimados: 1	**Puntos reales:** 1
Descripción: Mediante la programación en el script, crear objetos gráficos con valores definidos por el programador.	

Tabla 8 HU Crear objetos gráficos mediante scripting.

Historia de usuario	
Número: 7	**Nombre:** Adicionar objetos gráficos mediante scripting a los despliegues.
Usuario: Desarrollador	
Iteración asignada: 3	**Prioridad del Negocio:** Media
Puntos estimados: 1	**Puntos reales:** 1
Descripción: Mediante la programación en el script, agregar objetos gráficos al despliegue activo.	

Tabla 9 HU Adicionar objetos gráficos mediante scripting a los despliegues.

Historia de usuario	
Número: 8	**Nombre:** Salvar e importar scripts.
Usuario: Desarrollador	
Iteración asignada: 3	**Prioridad del Negocio:** Media
Puntos estimados: 1	**Puntos reales:** 1
Descripción: Permitir al editor de script, salvar el código script en ficheros de manera local así como importar archivos script salvados de manera local.	

Tabla 10 HU Salvar e importar scripts.

2.3.2 Diseño de Casos de Prueba.

A diferencia de las metodologías tradicionales, donde la fase de pruebas, incluyendo la definición de las mismas, generalmente se realiza al final del proyecto, o sobre el final del desarrollo de cada módulo; la metodología XP propone un modelo inverso, en el que, lo primero que se describe son las pruebas que el sistema debe pasar. (20)

- **Pruebas de aceptación.**

Las pruebas de aceptación son creadas en base a las historias de usuarios definidas por el cliente. Dichas pruebas son consideradas como pruebas de caja negra y los clientes son los responsables de verificar que el resultado de estas pruebas sean los correctos. Una HU puede tener más de una prueba de aceptación, tantas como sean necesarias para garantizar un correcto funcionamiento. (21)

Para la realización de las pruebas a la herramienta se diseñaron 7 casos de pruebas donde se ejecutaron paso a paso cada una de las posibles entradas al sistema y se evaluaron los resultados obtenidos de las mismas. Las tablas con los diseños de casos de prueba se colocaron en el próximo capítulo donde además se incluyeron los resultados obtenidos.

2.4 Fase de planificación.

La planificación es una fase corta donde el cliente estableció la prioridad de cada historia de usuario y correspondientemente, los programadores realizaron una estimación del esfuerzo necesario de cada una de ellas. Típicamente esta fase consiste en una o varias reuniones grupales de planificación y el resultado es un Plan de Entregas. Esta fase duró unos pocos días. Las estimaciones de esfuerzo asociado a la implementación de las historias la establecieron los programadores utilizando como medida el punto, lo que equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos. La planificación se puede realizar basándose en el tiempo o el alcance. Al planificar por tiempo, se multiplica el número de iteraciones por la velocidad del proyecto, determinándose cuantos puntos se pueden completar. Al planificar según el alcance, se divide la suma de puntos de las historias de usuario seleccionadas entre la velocidad del proyecto, obteniendo el número de iteraciones necesarias para su implementación. (16)

2.4.1 Estimación de esfuerzos por historia de usuario.

Para realizar un buen desarrollo del sistema propuesto, se realizó una estimación para cada una de las historias de usuario identificadas, llegando a los resultados que se muestran a continuación:

Historias de usuario	Puntos de estimación
Gestionar script.	2 Semanas
Asociar script a los recursos y controles.	1 Semana
Personalización de propiedades y animaciones.	1 Semana
Asociar variables a los componentes gráficos.	1 Semana
Crear objetos gráficos mediante scripting.	1 Semana
Adicionar objetos gráficos mediante scripting a los despliegues.	1 Semana
Salvar e importar scripts.	1 Semana
Depuración del código script.	1 Semana

Tabla 11 Estimación de esfuerzos.

2.4.2 Plan de iteraciones.

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. El plan de entrega está compuesto por iteraciones de no más de 3 semanas de duración. En la primera iteración se intentó establecer una arquitectura del sistema que pudiera ser utilizada durante el resto del proyecto; esto se logró estableciendo las historias de manera que forzaran la creación de la arquitectura antes mencionada. Al concluir la última iteración el sistema estuvo listo para entrar en producción.

Una vez definidas las HU y estimado el esfuerzo propuesto para la realización de cada una de ellas, se decidió realizar el sistema en 3 iteraciones, las cuales se describen a continuación de manera más detallada:

- **Iteración 1:**

Esta iteración tuvo como objetivo dar cumplimiento a las HU que se consideraron con una mayor importancia para el desarrollo de la herramienta. Al concluir dicha iteración se contó con todas las funcionalidades descritas en las HU 1 y 2, las cuales aluden la generación del código script y la asociación del mismo a los recursos y controles.

- **Iteración 2:**

Esta iteración tuvo como objetivo darle cumplimiento a las HU 3, 4 y 5, las cuales aluden la

personalización de propiedades y animaciones mediante el código script, la asociación de variables a los objetos gráficos y a la depuración del código para identificar posibles errores.

- **Iteración 3:**

Esta es la última iteración del mecanismo propuesto, en la cual se desarrollaron las HU 6 y 7, las cuales cumplen con la funcionalidad de la creación de objetos gráficos mediante el código script y su manejo en los despliegues, así como la de salvar e importar códigos script. Estas HU se integraron al resultado de las iteraciones anteriores para obtener la primera versión del sistema, a partir de este momento, el mecanismo se puso en un proceso de prueba para evaluar el desempeño del mismo.

2.4.3 Plan de duración de las iteraciones.

Como parte del ciclo de vida de un proyecto guiado por la metodología de desarrollo XP, se creó el plan de duración de cada una de las iteraciones que se llevaron a cabo durante el desarrollo del proyecto, su objetivo fundamental fue mostrar la duración de cada iteración, así como el orden en que serían implementadas las HU en cada una de las mismas según la prioridad asignada por el cliente.

Iteración	Historias de usuario	Duración total de las iteraciones
Iteración 1	Gestionar script.	3 Semanas
	Asociar script a los recursos y controles.	
Iteración 2	Personalización de propiedades y animaciones.	3 Semanas
	Asociar variables a los componentes gráficos.	
	Depuración del código script.	
Iteración 3	Crear objetos gráficos mediante scripting.	3 semanas
	Adicionar objetos gráficos mediante scripting a los despliegues.	
	Salvar e importar scripts.	

Tabla 12 Plan de duración de iteraciones.

2.5 Diseño de la solución propuesta.

La metodología XP, no requiere la descripción del sistema mediante diagramas de clase utilizando la notación UML, sino que en su lugar se guía por técnicas como las tarjetas CRC (Contenido, Responsabilidad y Colaboración). Esto no implica que no se utilicen dichos diagramas para obtener una mejor visión y comunicación entre el equipo de trabajo, siempre y cuando no posea una alta complejidad y defina información importante.

2.5.1 Tarjetas CRC.

Las características más sobresalientes de las tarjetas CRC son su simpleza y ductilidad. Una tarjeta CRC no es más que una ficha de papel o cartón que representa a una entidad del sistema. (22)

Las tarjetas CRC se utilizaron para estructurar las clases y a su vez definir las responsabilidades sobre las mismas, así como la simulación de escenarios en el sistema. A continuación se definen las tarjetas CRC del sistema:

Clase: Nombre de la clase que se está modelando	
Súper clase: Nombre de la clase padre en la herencia	
Sub Clase(s): Nombre de la(s) clase(s) hija en la herencia	
Responsabilidades: Es una descripción de alto nivel del propósito de la clase	**Colaboraciones:** Indica con cuáles otras clases se requiere relación para cumplir la responsabilidad

Tabla 13 Plantilla para las tarjetas CRC.

ScriptExtension	
Súper clase: -	
Sub Clase(s): ViewScriptExtension	
Responsabilidades: Interfaz abstracta necesaria para la creación de la extensión ViewScript, donde se manejará la gestión de los códigos script.	**Colaboraciones:** Clase QtPlugin, Clase QScriptEngine Clase HMI_Export

Tabla 14 Tarjeta CRC ScriptExtension.

ViewScriptExtension	
Súper clase: QObject, ViewExtension, ScriptExtension	
Sub Clase(s): -	
Responsabilidades: Clase que define la extensión ViewScript.	**Colaboraciones:** Clase QObject, Clase QScriptEngine Clase ViewExtension, Clase ViewScript Clase ScriptExtension, Clase VarIdClass

Tabla 15 Tarjeta CRC ViewScriptExtension.

CodeEditor	
Súper clase: QPlainTextEdit	
Sub Clase(s): -	
Responsabilidades: Clase que representa la zona de edición del código script.	**Colaboraciones:** Clase QPlainTextEdit, Clase QObject Clase QPaintEvent, Clase QResizeEvent Clase QSize, Clase QCompleter Clase QFocusEvent, Clase QKeyEvent Clase LineNumberArea, Clase QWidget Clase QAbstractItemModel

Tabla 16 Tarjeta CRC CodeEditor.

ViewScript	
Súper clase: ViewPluginInterface	
Sub Clase(s): -	
Responsabilidades: Clase que representa la interfaz de la extensión.	**Colaboraciones:** Clase QVBoxLayout, Clase ScriptEditor Clase ViewPluginInterface, Clase QWidget

Tabla 17 Tarjeta CRC ViewScript.

ScriptEditor	
Súper clase: QWidget	
Sub Clase(s): -	
Responsabilidades: Clase que representa la forma visual del editor mediante una ventana en la que estarán representadas las acciones a ejecutar sobre el código script y el editor de código.	**Colaboraciones:** Clase QWidget, Clase QMainWindow Clase QComboBox, Clase QToolButton Clase QGraphicsView, Clase HMICore Clase QScriptEngine, Clase CodeEditor Clase QScriptEngineDebugger Clase Highlighter

Tabla 18 Tarjeta CRC ScriptEditor.

LineNumberArea	
Súper clase: QWidget	
Sub Clase(s): -	
Responsabilidades: Clase que representa visualmente la zona de numeración de las líneas.	**Colaboraciones:** Clase QObject, Clase QPaintEvent Clase QResizeEvent, Clase QSize Clase QWidget, Clase QKeyEvent Clase CodeEditor

Tabla 19 Tarjeta CRC LineNumberArea.

Highlighter	
Súper clase: QSyntaxHighlighter	
Sub Clase(s): -	
Responsabilidades: Clase que representa visualmente las palabras reservadas de un color diferente.	**Colaboraciones:** Clase QSyntaxHighlighter, Clase QHash Clase QTextCharFormat, Clase QVector Clase QTextDocument, Clase QRegExp

Tabla 20 Tarjeta CRC Highlighter.

VarIdPrototype	
Súper clase: QObject, QScriptable	
Sub Clase(s): -	
Responsabilidades: Clase que permitirá crear prototipos de la clase VarId y acceder a todas sus propiedades.	**Colaboraciones:** Clase QtScript, Clase QObject Clase QScriptable, Clase VarId

Tabla 21 Tarjeta CRC VarIdPrototype.

VarIdClass	
Súper clase: QObject, QScriptClass	
Sub Clase(s): -	
Responsabilidades: Clase que permitirá manejar todo el acceso a las propiedades de la clase VarId.	**Colaboraciones:** Clase QtScript, Clase QObject, Clase VarId

Tabla 22 Tarjeta CRC VarIdClass.

2.5.2 Patrones de diseño.

Un patrón de diseño es un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la programación orientada a objetos. Existen tres clasificaciones para los mismos: Patrones de Creación, Patrones Estructurales y los Patrones de Comportamiento. (23)

Para la implementación del mecanismo en cuestión se utilizarán los patrones: Abstract Factory, Prototype.

Abstract Factory: Es un tipo de patrón creacional, que ofrece una interfaz para la creación de familias de productos relacionados o dependientes sin especificar las clases concretas a las que pertenecen. (24)

Prototype: Es un tipo de patrón creacional, que tiene como finalidad crear nuevos objetos duplicándolos, clonando una instancia creada previamente. Propone la creación de distintas variantes del objeto que la aplicación necesite, en el momento y contexto adecuado. (24)

2.5.3 Diagrama de Paquetes.

Un diagrama de paquetes es una colección de clases, relaciones, realizaciones de casos de uso, diagramas y otros paquetes que estén de alguna forma relacionados. Es utilizado para estructurar el modelo de diseño dividiéndolo en partes más pequeñas. Este diagrama se utilizó fundamentalmente como herramienta organizacional del modelo para agrupar los elementos relacionados con el mecanismo de scripting.

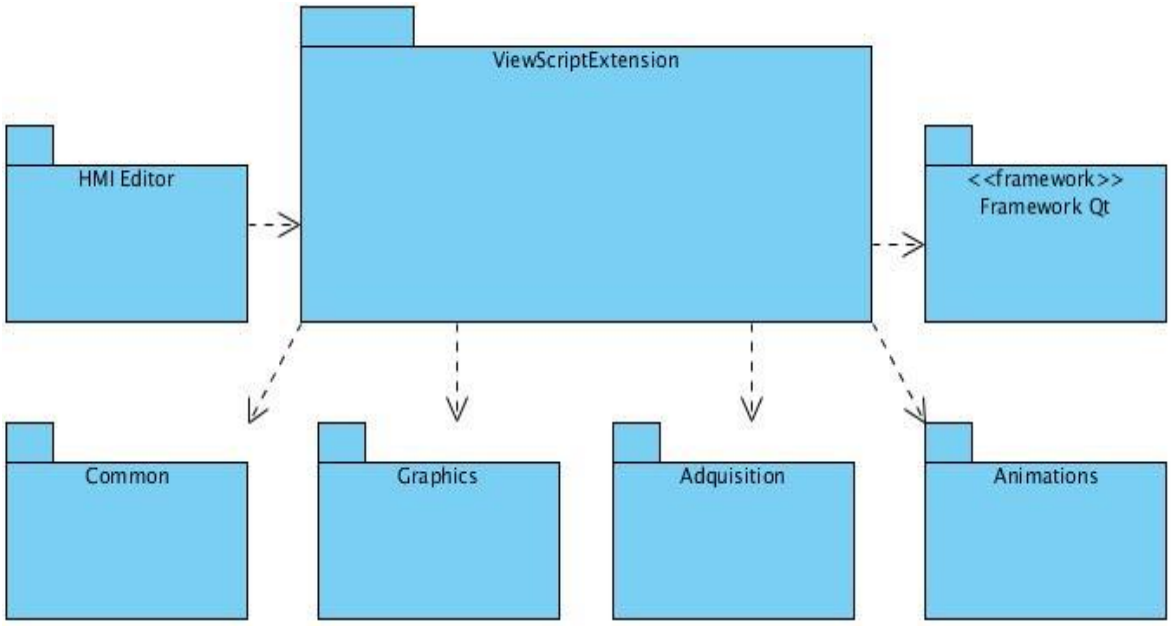

Ilustración 8 Diagrama de Paquetes.

2.5.4 Diagrama de Componentes.

La metodología XP plantea que para un mejor entendimiento de las tareas, flujos y métodos de desarrollo de las funcionalidades se pueden crear diagramas, siempre y cuando su creación no implique un mayor esfuerzo que la implementación del mismo. Siguiendo este principio se elaboró el diagrama de componentes, que muestra las dependencias lógicas entre componentes de software. Estos diagramas prevalecen en el campo de la arquitectura de software pero pueden ser utilizados para modelar y documentar cualquier arquitectura de sistema. (25)

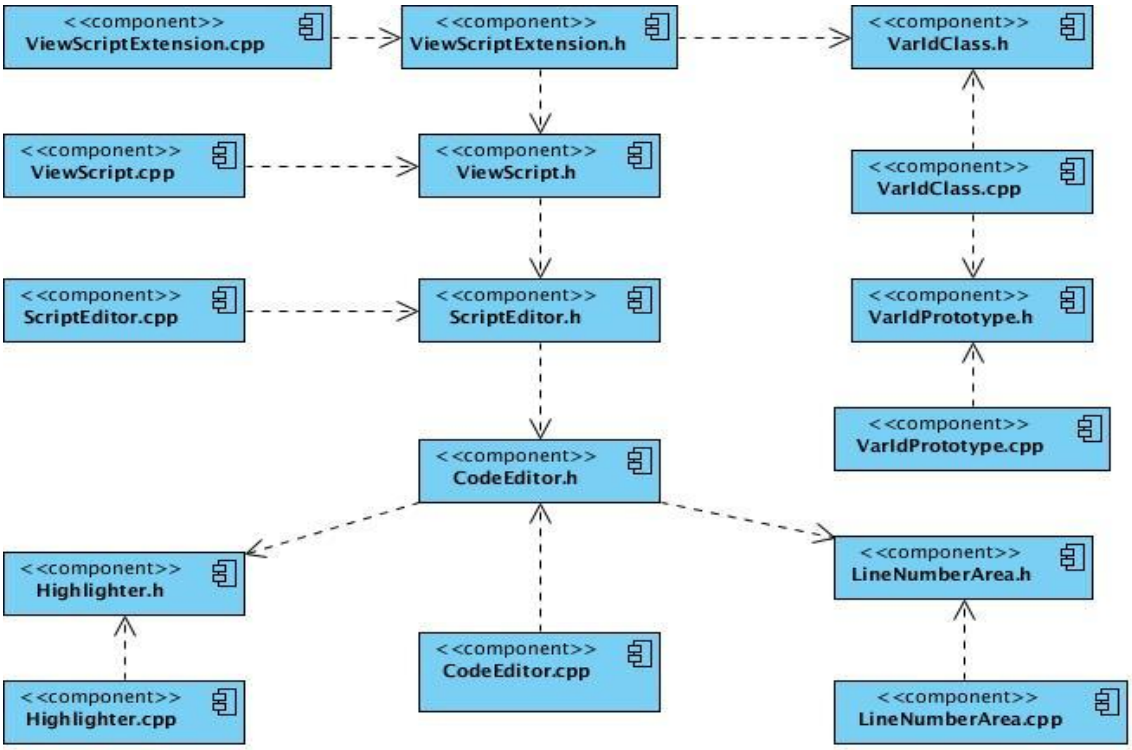

Ilustración 9 Diagrama de Componentes. Paquete ViewScriptExtension.

2.6 Consideraciones parciales

En este capítulo se comenzó el desarrollo de la propuesta del sistema, conociendo cómo se llevó a cabo el flujo actual de eventos. Para el cumplimiento del objetivo planteado se especificaron los roles de la metodología que estarán relacionados con el sistema. Además se realizó una descripción de las HU precisando por el cliente la prioridad de cada una de ellas, definiendo un orden lógico para la implementación de las mismas. Se definieron 8 HU que serán implementadas en 3 iteraciones, así como el modelo necesario para llevar a cabo la implementación del sistema. Además se definió por medio del diagrama de componentes las dependencias lógicas del sistema.

CAPÍTULO 3

IMPLEMENTACIÓN Y PRUEBA DEL MECANISMO DE SCRIPTING

3.1 Introducción

En el presente capítulo se detallaron las tres iteraciones llevadas a cabo durante la etapa de construcción del sistema, definiendo las tareas generadas a partir del desarrollo de las historias de usuario durante las iteraciones planificadas. Además quedaron especificados los resultados obtenidos de la ejecución de las pruebas de aceptación previamente diseñadas para probar las funcionalidades descritas y la valoración de dichas pruebas de manera cualitativa.

3.2 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 mantenible y que todos los participantes lo puedan entender en un menor tiempo. (26)

Para la implementación del mecanismo en cuestión es necesario utilizar el Estándar de codificación de C++ para el proyecto SCADA GALBA. (27)

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 hacer referencia a otras clases utilizar el comando @see.
- El código será escrito en inglés y la documentación en español.
- Las variables y funciones comienzan con letra minúscula. Cada palabra consecutiva en el nombre comienza con letra mayúscula.

3.3 Desarrollo de las iteraciones.

En la fase de Planificación se detallaron las historias de usuarios correspondientes a cada una de las iteraciones para desarrollar el sistema, teniendo presente las necesidades requeridas por el cliente. Durante el transcurso de cada iteración se realizó una revisión del plan de iteraciones. Como parte de este plan se desglosaron las historias definidas en tareas de programación o ingeniería, asignándole a un equipo de desarrollo o a una persona la responsabilidad de su implementación. Estas tareas son para el uso estricto del programador, por lo que no fue necesario escribirlas en un lenguaje no técnico para hacerla entendible al cliente.

A continuación se perfila con mayor detalle las tareas de desarrollo que se realizaron en cada una de las iteraciones:

3.3.1 Iteración #1

Esta iteración tuvo como objetivo dar cumplimiento a las HU que se consideraron con una mayor importancia para el desarrollo de la herramienta. Al concluir dicha iteración se contó con todas las funcionalidades descritas en las HU 1 y 2, las cuales aluden la generación del código script y la asociación del mismo a los recursos y controles.

Historias de usuario	Tiempo de implementación (semanas)	
	Estimación	Real
Gestionar script	2	2
Asociar script a los recursos y controles	1	1
Total	3	3

Tabla 23 Tiempo de implementación de la primera iteración.

Tarea de Ingeniería	
No. de la tarea: 1	**No. de la HU:** 1
Nombre de la tarea: Diseñar la interfaz de una extensión para el HMI que permita la gestión de scripts.	
Tipo de tarea: Diseño	**Puntos estimados:** 0.5
Fecha inicio: 14/05/2012	**Fecha fin:** 17/05/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Tiene como fin, crear la interfaz de la herramienta que se integrará al editor del HMI para la gestión de la lógica operacional.	

Tabla 24 Tarea #1 de la historia de usuario #1.

Tarea de Ingeniería	
No. de la tarea: 2	**No. de la HU:** 1
Nombre de la tarea: Implementar una extensión para el HMI que permita la gestión de scripts.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1.5
Fecha inicio: 17/05/2012	**Fecha fin:** 25/05/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Tiene como fin, dotar al editor del HMI de una herramienta para la gestión de la lógica operacional.	

Tabla 25 Tarea #2 de la historia de usuario #1.

Tarea de Ingeniería	
No. de la tarea: 3	**No. de la HU:** 2
Nombre de la tarea: Registro de los recursos y controles en el engine[12] de Qt Script.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 28/05/2012	**Fecha fin:** 01/06/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Tiene como fin, permitir al editor de script acceder a los recursos seleccionados para la asignación y ejecución del código implementado.	

Tabla 26 Tarea de la historia de usuario #2.

3.3.2 Iteración #2

Esta iteración tuvo como finalidad dar cumplimiento a las HU 3, 4 y 5, las cuales aluden la personalización

[12] Motor de ejecución del código script que provee Qt.

de propiedades y animaciones mediante el código script, la asociación de variables a los objetos gráficos y a la depuración del código para identificar posibles errores.

Historias de usuario	Tiempo de implementación	
	Estimación	Real
Personalización de propiedades y animaciones	1	1
Asociar variables a los componentes gráficos	1	1
Depuración del código script	1	1
Total	3	3

Tabla 27 Tiempo de implementación de la segunda iteración.

Tarea de Ingeniería	
No. de la tarea: 4	**No. de la HU:** 3, 6
Nombre de la tarea: Registro de los componentes gráficos en el engine de Qt Script.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 04/06/2012	**Fecha fin:** 22/06/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Tiene como fin, permitir al editor de script acceder a los metadatos[13] de los componentes gráficos.	

Tabla 28 Tarea de la historia de usuario #3 y #6.

Tarea de Ingeniería	
No. de la tarea: 5	**No. de la HU:** 4
Nombre de la tarea: Creación de una clase prototipo para el dato Varld.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 25/06/2012	**Fecha fin:** 27/06/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Tiene como fin, crear un prototipo de la clase Varld que permita el acceso de todas las propiedades de esta clase mediante el código script.	

Tabla 29 Tarea #1 de la historia de usuario #4.

[13] Datos estructurados y codificados que describen características de instancias conteniendo informaciones para ayudar a identificar, descubrir, valorar y administrar las instancias descritas.

Tarea de Ingeniería	
No. de la tarea: 6	**No. de la HU:** 4
Nombre de la tarea: Registro de la clase prototipo en el engine de Qt Script.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 27/06/2012	**Fecha fin:** 29/06/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Tiene como fin, permitir el acceso de la clase prototipo de VarId mediante el código script.	

Tabla 30 Tarea #2 de la historia de usuario #4.

Tarea de Ingeniería	
No. de la tarea: 7	**No. de la HU:** 5
Nombre de la tarea: Implementación de una sub-aplicación adjunta para la depuración.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 02/07/2012	**Fecha fin:** 06/07/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Tiene como fin, crear una ventana para ejecutar paso a paso el código script.	

Tabla 31 Tarea de la historia de usuario #5.

3.3.3 Iteración #3

Esta es la última iteración del mecanismo propuesto, en la cual se dió cumplimiento a las HU 6 y 7, las cuales cumplen con la funcionalidad de la creación de objetos gráficos mediante el código script y su manejo en los despliegues, así como la de salvar e importar códigos script.

Historias de usuario	Tiempo de implementación	
	Estimación	Real
Crear objetos gráficos mediante scripting.	1	1
Adicionar objetos gráficos mediante scripting a los despliegues.	1	1
Salvar e importar scripts.	1	1
Total	3	3

Tabla 32 Tiempo de implementación de la tercera iteración.

Tarea de Ingeniería	
No. de la tarea: 8	**No. de la HU:** 6
Nombre de la tarea: Crear un mecanismo para agregar objetos gráficos creados en el código script al despliegue activo.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 1
Fecha inicio: 10/09/2012	**Fecha fin:** 14/09/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Funcionalidad encargada de crear un mecanismo que permita al manejador crear componentes gráficos con valores personalizados mediante la programación script.	

Tabla 33 Tarea de la historia de usuario #6.

Tarea de Ingeniería	
No. de la tarea: 9	**No. de la HU:** 7
Nombre de la tarea: Crear un mecanismo para importar archivos de script al editor.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 17/09/2012	**Fecha fin:** 19/09/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Funcionalidad encargada de crear un mecanismo que permita al manejador importar archivos script en el editor.	

Tabla 34 Tarea #1 de la historia de usuario #7.

Tarea de Ingeniería	
No. de la tarea: 10	**No. de la HU:** 7
Nombre de la tarea: Crear un mecanismo para el almacenamiento del código script en archivos locales.	
Tipo de tarea: Desarrollo	**Puntos estimados:** 0.5
Fecha inicio: 19/09/2012	**Fecha fin:** 21/09/2012
Programador responsable: Yosvani Ramírez Martínez	
Descripción: Funcionalidad encargada de crear un mecanismo que permita al manejador salvar la programación creada en el editor en archivos locales.	

Tabla 35 Tarea #2 de la historia de usuario #7.

3.4 Pruebas.

Como define la metodología XP, uno de sus pilares es el proceso de pruebas. XP anima a probar tanto como sea posible. Esto permitió aumentar la calidad del sistema reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permitió aumentar la seguridad al evitar efectos colaterales no deseados realizando modificaciones y refactorizaciones. Esta metodología divide las pruebas del sistema en dos grupos: pruebas unitarias, encargadas de verificar el código y diseñada por los programadores y pruebas de aceptación o pruebas funcionales destinadas a evaluar si al final de cada iteración se consiguió la funcionalidad requerida por el cliente. (21)

3.4.1 Pruebas de aceptación.

Las pruebas de aceptación tienen mayor importancia que las pruebas unitarias, dado que significan la satisfacción del cliente con el producto desarrollado, el final de una iteración y el comienzo de la siguiente, por consecuente el cliente es la persona indicada para diseñar las pruebas a ejecutar. (21)

La ejecución de las pruebas previamente diseñadas, permitió la evaluación de las funcionalidades del mecanismo desarrollado antes de implantarlo en su entorno real de explotación. Los resultados obtenidos se muestran a continuación:

Caso de Prueba de Aceptación	
Número: 1	**Historia de usuario:** 1, 2, 3
Nombre: Crear script asociándolo a recursos y controles seleccionados.	
Descripción: Verificar la correcta asignación de un script al recurso o control seleccionado que modifique las propiedades del mismo.	
Condiciones de ejecución: El cliente debe comprobar que se pueda acceder a las propiedades del recurso o control, y mediante la ejecución del script, modificar las mismas.	
Entradas/Pasos de ejecución: Probar el acceso y modificación de las propiedades de cada uno de los recursos y controles del proyecto.	
Resultado esperado: Se accede a las propiedades de los recursos y controles del proyecto, modificando las mismas mediante scripting.	
Evaluación de la prueba: Satisfactorio	

Tabla 36 CP Crear script asociándolo a recursos y controles seleccionados.

Caso de Prueba de Aceptación	
Número: 2	**Historia de usuario:** 3
Nombre: Crear script asociándolo al componente gráfico seleccionado.	
Descripción: Verificar la correcta asignación de un script al componente gráfico seleccionado y la personalización de propiedades y animaciones del mismo mediante el script.	
Condiciones de ejecución: El cliente debe comprobar que se pueda acceder a las propiedades del componente gráfico, y mediante la ejecución del script, modificar las mismas y crear animaciones complejas utilizando estas.	
Entradas/Pasos de ejecución: Probar el acceso y modificación de las propiedades de cada uno de los componentes gráficos del proyecto y crear animaciones complejas utilizando estas en el script.	
Resultado esperado: Se accede a las propiedades de los componentes gráficos del proyecto, modificando las mismas mediante scripting y creando animaciones complejas utilizando estas.	
Evaluación de la prueba: Satisfactorio	

Tabla 37 CP Crear script asociándolo al componente gráfico seleccionado.

Caso de Prueba de Aceptación	
Número: 3	**Historia de usuario:** 4
Nombre: Asociar variable al componente gráfico seleccionado.	
Descripción: Verificar la correcta asignación de variables al componente gráfico seleccionado.	
Condiciones de ejecución: El cliente debe comprobar que se pueda acceder a las variables de los puntos de adquisición y realizar la asignación de las mismas al componente gráfico.	
Entradas/Pasos de ejecución: Probar el acceso y modificación de las variables de los puntos creados en adquisición y asignar las mismas a cada uno de los componentes gráficos del proyecto.	
Resultado esperado: Se accede a las variables de los puntos de adquisición y se realiza la asignación de estas a los componentes gráficos del proyecto.	
Evaluación de la prueba: Satisfactorio	

Tabla 38 CP Asociar variable al componente gráfico seleccionado.

Caso de Prueba de Aceptación	
Número: 4	**Historia de usuario:** 5
Nombre: Depuración del código script.	
Descripción: Verificar la correcta ejecución de la aplicación para la depuración del código script.	
Condiciones de ejecución: El cliente debe comprobar que se ejecute mediante una opción una aplicación para la depuración del código creado y así identificar posibles errores en el código.	
Entradas/Pasos de ejecución: Crear un código script y elegir la opción de depuración para identificar posibles errores.	
Resultado esperado: Se crea un código script con errores y se selecciona la opción de depuración, ejecutándose una ventana para la depuración del código.	
Evaluación de la prueba: Satisfactorio	

Tabla 39 CP Depuración del código script.

Caso de Prueba de Aceptación	
Número: 5	**Historia de usuario: 6**
Nombre: Creación de componentes gráficos mediante script y agregarlos al despliegue activo del proyecto.	
Descripción: Verificar la correcta creación de los componentes gráficos del sistema mediante script y su agregación al despliegue del proyecto activo.	
Condiciones de ejecución: El cliente debe comprobar que la herramienta permita crear componentes gráficos mediante la programación y agregarlos al despliegue activo del proyecto.	
Entradas/Pasos de ejecución: Crear un código script en el que se creen todos los componentes gráficos del sistema y agregarlos a un despliegue activo del proyecto.	
Resultado esperado: Se crean mediante el código script todos los componentes gráficos y se adicionan a un despliegue del proyecto.	
Evaluación de la prueba: Satisfactorio	

Tabla 40 CP Creación de componentes gráficos mediante script y agregarlos al despliegue activo del proyecto.

Caso de Prueba de Aceptación	
Número: 6	**Historia de usuario: 7**
Nombre: Importar códigos script desde archivos locales.	
Descripción: Verificar la correcta importación de códigos script al editor.	
Condiciones de ejecución: El cliente debe comprobar que la herramienta permita importar correctamente códigos script desde archivos locales.	
Entradas/Pasos de ejecución: Seleccionar la opción de abrir código existente y verificar la importación del archivo seleccionado.	
Resultado esperado: Se selecciona la opción de abrir y se importa el código script desde el fichero seleccionado.	
Evaluación de la prueba: Satisfactorio	

Tabla 41 CP Importar códigos script desde archivos locales.

Caso de Prueba de Aceptación	
Número: 7	**Historia de usuario: 7**
Nombre: Salvar códigos script en archivos locales.	
Descripción: Verificar el correcto almacenamiento de códigos script en ficheros.	
Condiciones de ejecución: El cliente debe comprobar que la herramienta permita almacenar correctamente códigos script en archivos locales.	
Entradas/Pasos de ejecución: Seleccionar la opción de salvar el código creado y verificar el almacenamiento en el archivo seleccionado.	
Resultado esperado: Se selecciona la opción de salvar y se crea un fichero con el código script en caso de seleccionar un archivo inexistente.	
Evaluación de la prueba: Satisfactorio	

Tabla 42 CP Salvar códigos script en archivos locales.

3.5 Resultados del mecanismo de integración de scripting sobre el HMI del SCADA GALBA.

Después de integrado el mecanismo para optimizar las configuraciones realizadas en el HMI del SCADA GALBA, los operadores se vieron beneficiados con el uso de la herramienta de gestión de la lógica operacional generada por la extensión ViewScriptExtension, disminuyendo la complejidad de realizar de forma visual las configuraciones en el editor. Se logró una integración entre los diferentes componentes de una manera más sencilla, permitiendo la creación de aplicaciones más completas y complejas.

Ilustración 10 Editor de script del SCADA GALBA.

3.6 Consideraciones parciales

En este capítulo se definió el estándar de codificación a utilizar. Se desarrollaron las tareas correspondientes para dar solución a las historias de usuario. Además se ejecutaron las pruebas de aceptación diseñadas en el capítulo anterior. A partir del resultado arrojado por las pruebas realizadas se llegó a la conclusión de que el mecanismo se encuentra listo para su puesta en funcionamiento. Con la finalización de este capítulo se da por terminada la propuesta de solución del mecanismo a elaborar por el autor.

CONCLUSIONES GENERALES

El presente trabajo tuvo como base la investigación y el desarrollo científico encaminados a la elaboración e integración de un mecanismo de programación script al módulo HMI del SCADA GALBA, para elevar las funcionalidades de configuración en dicho módulo. En el mismo se realizó un estudio de aplicaciones similares y lenguajes scripting utilizados por estas, logrando adquirir un conocimiento de las funcionalidades que serían necesarias implementar.

Como resultado del trabajo realizado se logró la implementación de un mecanismo que brinda las siguientes ventajas:

- Flexibilidad y dinamismo en las configuraciones realizadas en el EC.
- Personalización de propiedades de los objetos.
- Utilización de funciones matemáticas para el cálculo y ejecución de animaciones.
- Permite definir de manera ordenada secuencias de animaciones.
- Permite elevar el nivel de configuración de los eventos asociados a los componentes gráficos.
- Permite el vínculo con aplicaciones externas para la gestión de códigos script.
- Fácil uso de la aplicación.
- Generación de plantillas de scripting.
- Reutilización del código de la herramienta para aplicaciones con características similares.

El estudio en las diferentes temáticas en el campo de las herramientas gráficas para la gestión de la lógica operacional en un sistema de supervisión y control, se ven reflejado en el resultado alcanzado.

RECOMENDACIONES

Los objetivos planteados para el desarrollo de este trabajo fueron alcanzados, aunque se considera necesario continuar su perfeccionamiento con vistas a incrementar las prestaciones del mecanismo. A continuación se reflejan algunas recomendaciones que se consideraron importantes para futuras versiones:

- Mejorar la herramienta auxiliar de depuración del código.
- Disponer de un conjunto de instrucciones con métodos de funciones personalizados.
- Elevar el nivel de configuración de las alarmas a partir del trabajo con scripting.
- Explotar la utilización de esta tecnología para lograr un mejor funcionamiento del sistema.

REFERENCIAS BIBLIOGRÁFICAS

1. **Montero, Dagoberto, Barrantes, David B. y Quirós, José M.** Introducción a los sistemas de control supervisor y de adquisición de datos (SCADA). [En línea] 2004. [Citado el: 5 de Septiembre de 2012.] http://es.scribd.com/doc/13473499/Introduccion-a-Los-Sistemas-SCADA.

2. **Mandado, Enrique.** *Autómatas programables y sistemas de automatización.* Barcelona : 2009.

3. **EcuRed.** Script. [En línea] 2011. [Citado el: 20 de Sptiembre de 2012.] http://www.ecured.cu/index.php/Script.

4. **Martínez, Lourdes, Santos, Hanoi y Otero, Eduardo.** Impacto de algunas tecnologías en el desarrollo de los sistemas SCADA. [En línea] 2005. [Citado el: 7 de Septiembre de 2012.] http://www.scielo.org.ve/scielo.php?script=sci_arttext&pid=S131648212005000400008&lng=pt&nrm=iso&tlng=pt.

5. **ISaGRAF Enhanced.** HMI Scripting Language Reference. [En línea] 2004. [Citado el: 10 de Septiembre de 2012.] http://www.isagraf.com/pages/.../hmi_scripting.pdf.

6. **Grant, Calum.** C++ Script a scripting language built on C++. [En línea] 2008. [Citado el: 14 de Septiembre de 2012.] http://calumgrant.net/cppscript/index.html.

7. **ECMA International.** *ECMAScript Language Specification. ECMA International* : Geneva, 2011.

8. **Qt Project.** Qt Documentation Snapshots. [En línea] 2012. [Citado el: 16 de Septiembre de 2012.] http://doc-snapshot.qt-project.org/5.0/qtscript/qtscript-index.html

9. **Invensys Systems.** InTouch HMI Scripting And Logic Guide. [En línea] 2007. [Citado el: 25 de Septiembre de 2012.]

10. **OMRON.** CX-Supervisor. User Manual. Software Release 3.1. [En línea] 2012. [Citado el: 4 de Octubre de 2012.]

11. **SIMATIC WinCC.** Manual de WinCC V 6.0. [En línea] 2010. [Citado el: 22 de Noviembre de 2012.] http://es.scribd.com/doc/57325173/Curso-WinCC-V6.

12. **MOVICON X1.** Programmer's manual. [En línea] 2006. [Citado el: 22 de Noviembre de 2012.]

13. **Fundación Eclipse.** Eclipse DemoCamps Juno 2012. [En línea] 2012. [Citado el: 28 de Noviembre de 2012.] http://www.eclipse.org/

14. **Stroustrup, Bjarne.** *The C++ Programming Language (Third Edition):* Addison-Wesley, 1997.

15. **Qt Project.** Qt Project. [En línea] 2012. [Citado el: 3 de Diciembre de 2012.] http://qt-project.org/wiki/Wiki_Home_Spanish

16. **Letelier, Patricio y Penadés, Ma. Carmen.** *Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP).*

17. **Calero Solís, Manuel.** *Una explicación de la programación extrema (XP).* 2003.

18. **UML.** Object Management Group – UML. [En línea] 2013. [Citado el: 8 de Enero de 2013.] http://www.uml.org/

19. **Visual Paradigm.** UML CASE tool for software development. [En línea] 2013. [Citado el: 10 de Enero de 2013.] http://www.visual-paradigm.com/product/vpuml/

20. **Joskowicz, José.** Reglas y Prácticas en eXtreme Programming. [En línea] 2008. [Citado el: 3 de Febrero de 2013.] http://www.google.com.cu/url?sa=t&rct=j&q=reglas+y+pr%C3%A1cticas+en+extreme+programming&source=web&cd=1&ved=0CCwQFjAA&url=http%3A%2F%2Fwww.uls.edu.sv%2Findex.php%3Foption%3Dcom_phocadownload%26view%3Dcategory%26download%3D90%3Areglas-y-prcticas-en-progra.

21. **Gutiérrez, Javier J., Escalona, M. J., Mejías, M. y Torres, J.** Pruebas del sistema en Programación Extrema. [En línea] 2006. [Citado el: 12 de Febrero de 2013.] http://www.lsi.us.es/~javierj/investigacion_ficheros/PSISEXTREMA.pdf

22. **Casas, Sandra y Reinaga, Héctor.** Identificación y Modelado de Aspectos Tempranos dirijido por Tarjetas de Responsabilidades y Colaboraciones. [En línea] 2008. [Citado el: 4 de Febrero de 2013.] http://sedici.unlp.edu.ar:8080/bitstream/handle/10915/21813/Documento_completo.pdf?sequence=1.

23. **Escuela politécnica Superior.** *Patrones de diseño.* [En línea] 2008. [Citado el: 6 de Febrero de 2013.]

24. **Cooper, James W.** *Java™ Design Patterns: A Tutorial:* Addison-Wesley, 2000.

25. **Arizaca Ramírez, Elisa.** Artefacto: Diagrama de componentes. [En línea] 2009. [Citado el: 6 de Febrero de 2013.] http://virtual.usalesiana.edu.bo/web/practica/archiv/compon.doc.

26. **Arias Calleja, Manuel.** Carmen. Estándares de codificación. [En línea] 2006. [Citado el: 7 de Febrero de 2013.] http://www.chubut.gov.ar/informatica/docs/EstandaresCodificacion.pdf.

27. **Chávez Lorenzo, Ariel.** *Estándares de codificación para C++.* 2011.

GLOSARIO DE TÉRMINOS

ActiveX: Entorno para definir componentes de software reusables de forma independiente del lenguaje de programación.

Amigable: En términos informáticos se refiere a la facilidad de uso o sencillez de una aplicación.

API: Interfaz de programación de aplicaciones.

Despliegue: Vista que se desea supervisar de la industria.

Depuración: Es el proceso de identificar y corregir errores en la programación.

Engine: Motor de ejecución del código script que provee Qt.

Estándar: Norma que se utiliza como punto de partida para el desarrollo de servicios, aplicaciones, protocolos, entre otros.

Framework: En el desarrollo de software es una estructura de soporte definida 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 software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Metadatos: Datos estructurados y codificados que describen características de instancias conteniendo informaciones para ayudar a identificar, descubrir, valorar y administrar las instancias descritas.

Middleware: Software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

SCADA: Sistema para la automatización de procesos industriales que permite la supervisión y el control de dichos procesos.

Software: Es la suma total de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de cómputo.

Shells: Intérprete de órdenes, son aplicaciones capaces de interpretar las órdenes del usuario a través de comandos escritos, como por ejemplo el sistema MS-DOS o los terminales de consola de los sistemas operativos Linux. Estas aplicaciones permiten al usuario interactuar con el ordenador, normalmente a través de una sencilla interfaz de texto plano y suponen la forma más básica de interacción de un usuario con su ordenador, escribiendo las órdenes en este Shell a través de comandos y recogiendo las respuestas de la máquina.

Runtime: Sub-modulo de ejecución en tiempo real de la aplicación.

Ventana: Parte fundamental de la interfaz gráfica de la computadora, especialmente con sistemas operativos que permiten el multiprocesamiento.