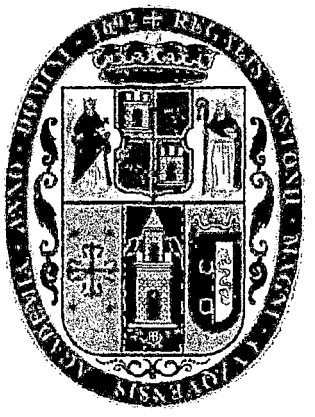
UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE CIENCIAS QUÍMICAS, FÍSICAS, MATEMÁTICAS, FARMÁCIA E INFORMÁTICA

CARRERA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS



"DISEÑO E IMPLEMENTACIÓN DE COMPONENTES DE SOFTWARE, PARA EL DESARROLLO DE APLICACIONES SCADA"

Tesis Presentada Por:

Br. Darío Cesar Salcedo Moreno
Br. Albert Bayro Alvarez

Para optar al Título profesional de:

Ingeniero Informático Y De Sistemas

Asesor:

Ing. Edwin Carrasco Poblete

CUSCO – PERÚ
2013

Declaración

Nosotros **Darío Cesar Salcedo Moreno y Albert Bayro Alvarez,** declaramos bajo juramento que el tema de tesis aquí descrito es de nuestra autoría, que no ha sido previamente presentada para ningún grado o calificación profesional en esta Universidad o en otras del país u otros lugares y se ha consultado las referencias bibliográficas y recorrido la web para cercioramos de que es cierta la información que damos en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Universidad Nacional De San Antonio Abad Del Cusco (UNSAAC), según lo establecido por la ley de propiedad intelectual, por su reglamento y por la normatividad institucional vigente.

DARIO CESAR SALCEDO MORENO
DNI 42128062

ALBERT BAYRO ALVAREZ
DNI 43606792

Certificación

Certifico que el presente trabajo fue desarrollado por Darío Cesar Salcedo Moreno y Albert Bayro Alvarez, bajo mi supervisión

Ing. Edwin Carrasco Poblete

Asesor Del Proyecto

Agradecimientos

A Dios

Al padre eterno, sin nombre y sin fin, por permitirme seguir día a día en pie luchando, pues todo lo que soy se lo debo a su gracia y misericordia.

A Mis Padres

A Manuel y Luisa, por su infinito amor y apoyo incondicional a lo largo de mi vida, a su ejemplo de sacrificio, constancia, fortaleza y valores morales.

Por todas las privaciones, tristezas y desvelos por los cuales tuvieron que pasar, para brindar a sus hijos una educación no solamente escolar sino personal y universitaria.

En recuerdo a los lomos cansados de mis padres, como raíces cuyos tallos no se torcieron. En recuerdo de aquel llanto, cuyas lágrimas mojaron las mejillas del hijo enfermo.

A Mi Familia y Amigos

A mis hermanos Rosita, Angélica y Edwin por compartir junto a mí cada una de las vivencias y experiencias de mi vida y por estar siempre dispuestos a poyarme en cualquier circunstancia.

A mis grandes amigos Emerson Pastor Chambi y Jefferson Olivera Altamirano, por compartir triunfos, alegrías, tristezas y sufrimientos a lo largo de mi vida y etapa universitaria.

A todos quienes estuvieron en algún momento de mi vida para darme su apoyo. A mis demás amigos que siempre están "molestando", apoyando y brindando alegría a la vida.

Darío Cesar.

Agradecimientos

A Dios

Por haberme permitido llegar hasta este punto, de poder vivir la realidad de este sueño anhelado y haberme dado salud para lograr mis objetivos, además de su infinita bondad y amor.

A Mi Familia

Quienes no se cansan de darme todo su apoyo en las buenas en las malas, me dan las fuerzas necesarias para que cada día intente superarme como persona, el esfuerzo de mis padres y de mis hermanos reflejan el momento hoy vivido por mi persona.

A Mis Maestros

Por sus enseñanzas y concejos que nos dieron para la culminación de nuestros estudios profesionales apoyos, motivaciones y el tiempo compartido para que este trabajo de tesis sea elaborado de la manera más profesional posible además por impulsar el desarrollo de nuestra formación profesional.

A mis Amigos

Que nos apoyábamos mutuamente en nuestra formación profesional y que hasta ahora, seguimos siendo amigos: Cesar mi compañero de tesis con quien dimos todo para hacer este trabajo, pasando una y otras adversidades que pudimos sacar adelante, las enseñanzas, los consejos, el aliento que me dio sirvió para poder aprender más.

Albert

DEDICATORIA

Para mí el haber logrado obtener este título significa que he logrado armar el Rompecabezas de la llave que abre la puerta a nuevas oportunidades y un mejor porvenir, dándole a mi vida un brillo que ilumina a todos aquellos que me rodean.

El camino de la vida no siempre está lleno de triunfos y glorias ni tampoco sería transitable sin la ayuda de pequeños ángeles que nos alientan, apoyan y enseñan.

Dedico este pequeño triunfo, mi primer sueño cumplido, que parecía inalcanzable a toda mi familia, y en especial a mis padres, que han sido mis ángeles en la vida.

Este logro es por y para ustedes

Darío Cesar.

DEDICATORIA

Dedico este paso importante en mi vida a mi madre que ha sabido formarme con buenos sentimientos, hábitos y valores, lo cual me ha ayudado a salir adelante en los momentos más difíciles.

A mi padre quien con sus consejos ha sabido guiarme para conseguir este objetivo, sin importarle los sacrificios que en su momento hizo para que pueda llegar este momento tan feliz para nosotros.

A mis hermanos quienes a cada momento me hacían recordar quien era y para que había estudiado, dándome ánimos y fuerzas para poder conseguir este triunfo.

A mis maestros y a todas las personas que nos bridaron su ayuda en este proyecto.

Este logro es para ustedes

Albert

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

Después de haber sido desarrolladores durante muchos años, hemos alcanzado el punto en el que programar empezaba a cansarnos y aburrirnos, perdiendo poco a poco el disfrute que nos causaba implementar aplicaciones, esto debido entre otras cosas porque las herramientas de las que disponíamos nos hacían pensar en que ya se hizo todo, no hay mucho que explorar o porque los planes que teníamos no los podíamos llevar a cabo a causa de sus limitaciones. Entonces llego Microsoft Visual Studio 2008 con su tecnología WPF y su suite de nuevas y excitantes herramientas de Microsoft Expression Studio, que abrió todo un mundo de posibilidades, un nuevo despertar que por fin resuelve cosas con las que hemos estado luchando a brazo partido por mucho tiempo.

En lo referente a empresas industriales, cabe destacar que uno de los objetivos es optimizar los sistemas de control, el cual les permita la utilización eficiente de los recursos, reduciendo tiempos de preparación, minimizando errores, eliminando tiempos de espera, logrando un flujo continuo en la línea de producción, minimizando los transportes internos, evitando retrasos, etc. Ofrecer entonces un sistema de información a estas empresas, el cual les ayude a lograr sus objetivos es una tarea esencial, pero el mero hecho de intentar abordar este problema no resultaba sencillo debido a las limitaciones innatas con las que cuentan los lenguajes de programación tradicionales.

Desarrollar un sistema de información para visualizar la evolución de un proceso o averiguar el estado de una variable como por ejemplo el caudal de una mezcla, implicaba tener que construir un control de usuario para llevar a cabo esta función, lo cual cabe indicar que era una tarea muy compleja que implicaba tener un profundo conocimiento sobre el sistema grafico del lenguaje y tener mucho tiempo disponible, para desarrollar aplicaciones con ellos, tarea que al final muchas veces no siempre resultaba satisfactoria.

Esta problemática, unida a las ideas que teníamos rezagadas durante mucho tiempo, por fin pueden ser abordadas con relativa facilidad, haciendo uso de WPF y el paquete Expression Studio, el cual nos devuelve la emoción de escribir código otra vez, nos hace levantar cada mañana pensando en todo lo nuevo y fabuloso que aprenderemos en el día. Este proyecto de titulación surgió entonces como un reto para poner en práctica nuestra habilidad y conocimientos, descubrir la potencia de estas herramientas y sobre todo ofrecer a la comunidad de desarrolladores, componentes que faciliten la implementación de sistemas de información industriales, que proporcionen una ventaja competitiva frente a la constante y dura

competencia que hoy en día poseen todas las empresas, dotándolas de información veraz y oportuna para la toma de decisiones.

El resultado de este proyecto será una herramienta eficaz y oportuna que permitirá a los desarrolladores de aplicaciones orientados al campo industrial, realizar su trabajo de manera más fácil y segura en base a los controles que ofrece el proyecto, optimizando tiempos de desarrollo y porque no costos.

ABSTRACT

Having been developers for many years, we have reached the point where the programing was starting to get tired and bored, slowly losing the enjoyment that caused us to deploy applications, this due not least because the tools available to us made us think they already did everything, not much to explore or because the plans we had we could not perform because of their limitations. Then came Microsoft Visual Studio 2008 with WPF technology and exciting suite of new tools Microsoft Expression Studio, which opened up a whole world of possibilities, a new awakening to finally resolve things with which we have been fighting tooth and nail for long time.

With regard to industrial, note that one of the objectives is to optimize the control systems, which allowed the efficient use of resources, reducing setup times, minimizing errors, eliminating waiting times, achieving a continuous flow the production line, minimizing internal transport, avoiding delays etc. Then provide an information system for these companies, which will help them achieve their goals is an essential task, but the mere fact of addressing this problem was not easy because of the inherent limitations that come with traditional programming languages.

Develop an information system to visualize the evolution of a process or determine the status of a variable such as the flow of a mixture meant having to build a user control to carry out this function, which should be noted that it was a very complex task that involved have a deep knowledge of the system graphic language and have plenty of time to develop applications with them, a task that often end was not always satisfactory.

This problem, together with the ideas we had left behind for a long time, finally can be addressed relatively easily, using WPF and Expression Studio package, which returns the thrill of writing code again, makes us lift each morning thinking about what we will learn fabulous new day. This degree project emerged then as a challenge to implement our expertise and knowledge, discover the power of these tools and above all provide the community of developers, components that facilitate the implementation of industrial information systems that provide a competitive advantage constant against tough competition today have all companies, providing them with accurate and timely information for decision-making.

The result of this project will be a timely and effective tool that will enable application developers oriented industrial field, make their jobs easier and safer controls based on offering the project, optimizing development time and because not costs.

PALABRAS CLAVE

Aplicaciones SCADA, Controles Industriales, Sistemas de Control, Control de Usuario, Procesos Industriales, Interfaz Hombre Maquina, Componentes de Software, Controles Personalizados.

KEYWORDS

SCADA Applications, Industrial Controls, Control Systems, User Control, Industrial Processes, Human Machine Interface, Software Components, Custom Controls.

PRESENTACIÓN

Señor Decano de la Facultad de Ciencias Químicas, Físicas y Matemáticas de la Universidad Nacional de San Antonio Abad del Cusco.

Distinguidos Señores miembros del Jurado:

Al término de nuestros estudios profesionales en Ingeniería Informática y de Sistemas, conforme exige el reglamento de Grados y Títulos de la Facultad de Ciencias Químicas, Físicas y Matemáticas, ponemos a vuestra consideración el presente trabajo de tesis intitulado: "Diseño e implementación de componentes de software, para el desarrollo de aplicaciones SCADA ", con la finalidad de otorgar a los desarrolladores de software, herramientas que agilicen la construcción de aplicaciones industriales.

Esperando que los miembros del jurado eximan las deficiencias que pudieran presentarse y valoren el contenido desarrollado que trata de aportar con el estudio y desarrollo de nuestra sociedad. Sin más, los invitamos a revisar la presente documentación que esperamos satisfaga sus expectativas.

Los Autores

CAPÍTULO 1

ASPECTOS GENERALES

En el presente capítulo se dará a conocer la definición del problema, justificación, limitaciones, metodología, entre otros.

1.1 DESCRIPCIÓN GENERAL DEL PROBLEMA

Las empresas industriales en nuestro país han ido haciendo implantaciones de tecnología de automatización en mayor o menor grado, de acuerdo al sector en el que se desenvuelven. A causa de esto se dispone de mucha información por cada proceso productivo, pero ésta se encuentra dispersa y desordenada por lo que no resulta útil al momento de hacer un análisis para poder tomar decisiones dentro del proceso productivo.

La importancia de mejorar los procesos productivos es el objetivo principal a conseguir en cualquier industria, el cual les permita la utilización eficiente de los recursos, reduciendo tiempos de preparación, minimizando errores, eliminando tiempos de espera logrando un flujo continuo en la línea de producción, minimizando los transportes internos, evitando retrasos, etc.

Surge la necesidad entonces de ofrecer herramientas tecnológicas y sistemas de control que permitan por ejemplo:

- Tomar mejores decisiones acerca de la producción, montando sensores y operadores en puntos críticos de un proceso productivo, de tal manera que se podría proporcionar una vista detallada de la operación al momento en que éstas ocurran, así se podrán corregir desviaciones en el momento oportuno.

- Aligerar las tareas de mando y supervisión, con un control más sofisticado, complementándose con sistemas de comunicación y herramientas de visualización que permitan efectivizar este proceso.

- Aprovechar al máximo la información que se obtiene día a día de los diversos procesos y así poder optimizar el uso de maquinaria, energía, mano de obra y en general todos los recursos invertidos para la elaboración de un producto determinado, de esta manera se podría garantizar que se cumplan los periodos de tiempo establecidos para elaborar los productos, la calidad de éstos y mejorar las utilidades de una empresa cumpliendo e incluso superando las expectativas de sus clientes.

Al momento de abordar este problema para intentar solucionar las deficiencias con las que cuenta una industria, los desarrolladores se encuentran en un gran dilema al no contar con herramientas que le faciliten el desarrollo de aplicaciones SCADA. Lo cual hace crecer el costo y tiempo de desarrollo del software al tener que adquirir librerías de terceros fabricantes o invertir tiempo y dinero en la construcción de sus propios controles.

Otro escenario es que sencillamente podrían dejar de lado ese proyecto para dedicarse a la construcción de software convencional (Logística, Gestión, etc.), aumentando la competencia en sectores por demás sobresaturado.

1.2 IDENTIFICACIÓN DEL PROBLEMA

La implementación de sistemas de información orientadas al sector industrial requiere de mayor tiempo y costo debido a que los lenguajes de programación cuentan con controles convencionales y no se dispone de componentes que permitan manipular variables de tipo industrial.

1.3 ANTECEDENTES

Vista la necesidad de tener que liberar al operario de su función de actuación física directa en la planta reduciendo su participación en áreas críticas, de tener un control más preciso y agudo de las variables de producción y de contar con información relevante de los distintos procesos en tiempo real para lograr mejoras significativas en la gestión; varios fabricantes desarrollaron desde entonces paquetes de software capaces de comunicarse con los sistemas de control existentes. Esta tendencia ha ido en aumento hoy en día, de tal manera que las opciones existentes en el mercado son numerosas.

Actualmente, los fabricantes de software industrial ponen a disposición del mercado un amplio abanico de sistemas HMI, orientados a satisfacer distintos niveles de necesidades.

De esta manera, están los HMI básicos (desarrollados a medida en un entorno de programación), que pueden ser simplemente una interfaz de panel con una mínima cantidad de data, pero también hay soluciones integrales (paquetes enlatados HMI, que contemplan la mayoría de las funciones estándares de los sistemas SCADA), a las que pueden agregárseles herramientas de espectro más amplio, lo cual ha hecho que la mayoría de las empresas, sin importar su tamaño, no conciban sus procesos sin herramientas de este tipo.

Haremos referencia a las herramientas de desarrollo de sistemas SCADA más conocidos con las que se cuenta en la actualidad, dando una pequeña descripción sobre ellos:

1.3.1 EMPRESAS QUE DESARROLLAN COMPONENTES

Entre las compañías de fama mundial que desarrollan y personalizan componentes para la plataforma .Net y otros lenguajes de programación, de entre las cuales algunos de sus productos están orientados al sector industrial, podemos citar a los siguientes:

- DevExpress UI Controls de la empresa DevExpress
- DotNetBar de la empresa DevComponents
- DotNetMagic de la empresa Crownwood Software
- Essential Studio de la empresa SyncFusion
- Ingragistics NetAdvantage de la empresa Global Software Company
- Krypton Toolkit de la empresa ComponentFactory
- RadControls de la empresa Telerik

Estas empresas ofrecen un conjunto de herramientas de desarrollo de interfaz de usuario y componentes para capa de presentación disponible para Windows y otras plataformas.

Sus productos permiten a los desarrolladores crear interfaces de usuario, que son la base para el desarrollo de todo tipo aplicaciones de negocio, visualización de datos en línea y aplicaciones industriales; para las plataformas que incluyen Windows Forms, Windows Presentation Foundation (WPF), ASP.NET y Silverlight, así como jQuery / HTML5 y controles móviles, para el Windows Phone, iOS (iPhone y iPad) y Android.

Estos productos incluyen todos los elementos más importantes de interfaz, como grids, agendas, gráficos, barras de herramientas, menús, listas, árboles, pestañas, barras de navegación y algunos controles industriales, de los cuales mostramos unos ejemplos a continuación:

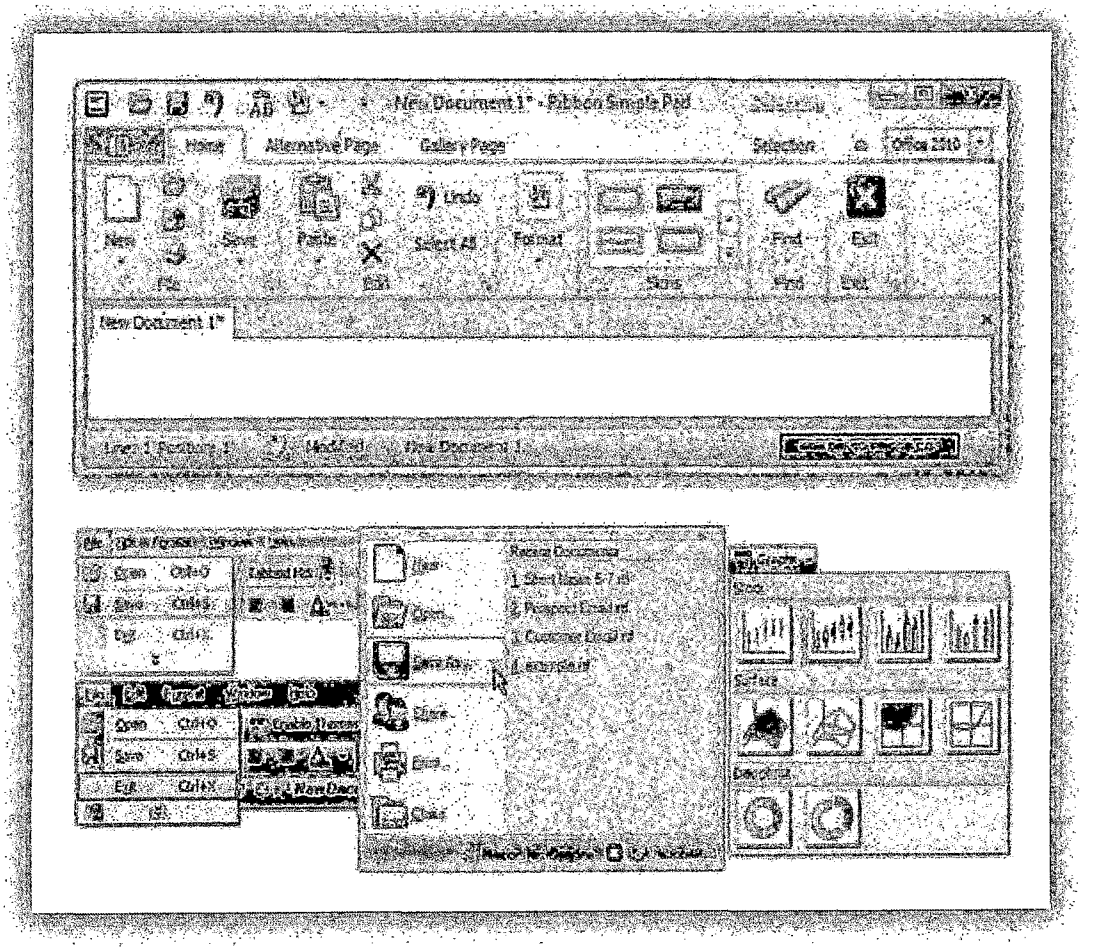


Figura 1: Controles Personalizados que ofrecen estas empresas
Fuente: http://www.devcomponents.com/dotnetbar/
Acceso: 15 de Diciembre del 2012

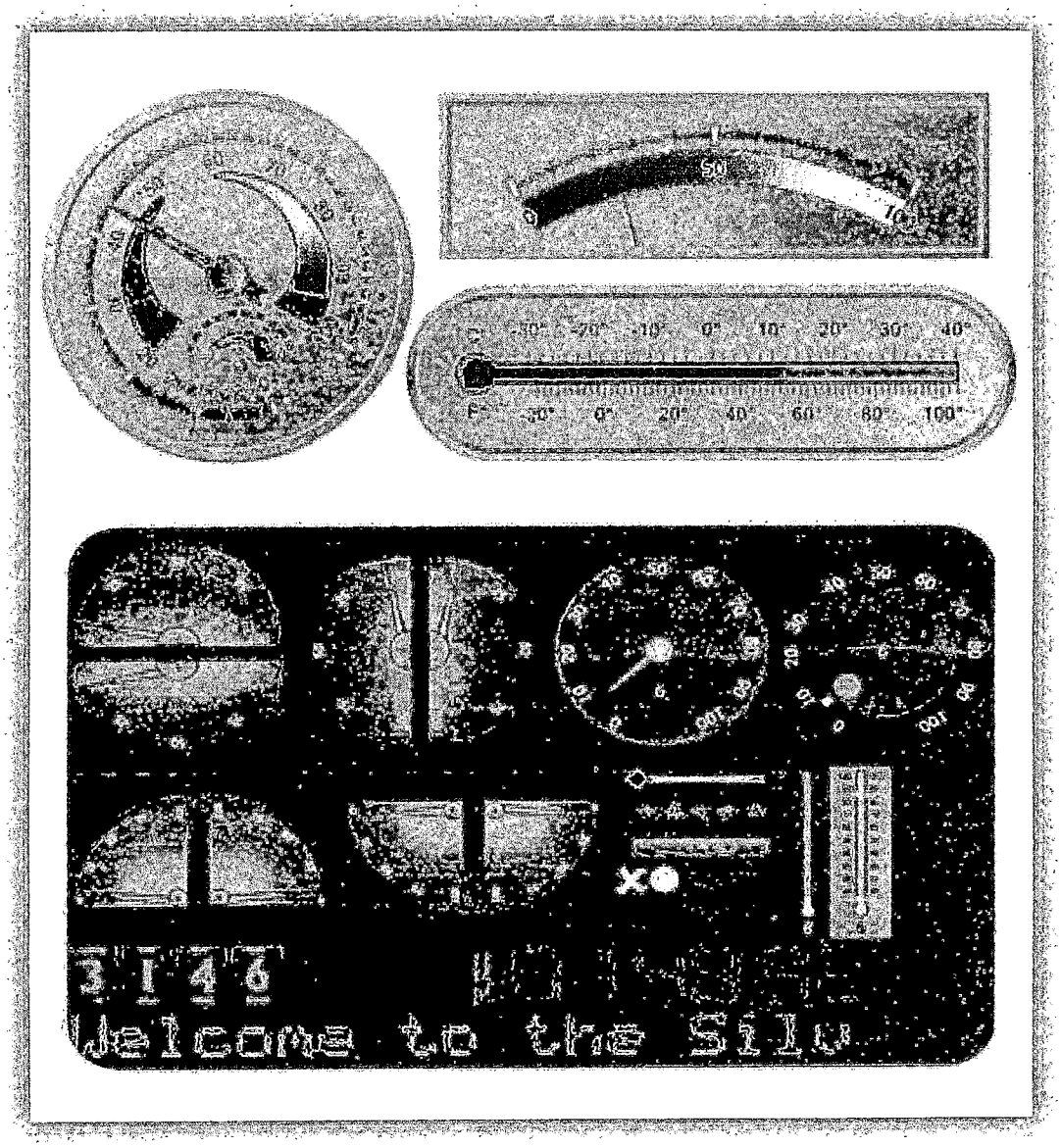


Figura 2: Controles industriales que ofrecen estas empresas
Fuente: http://www.devexpress.com/Products/NET/Controls
Acceso: 15 de Diciembre del 2012

1.3.2 EMPRESAS QUE OFRECEN ENTORNOS DE DESARROLLO PARA LA CONSTRUCCION DE SISTEMAS SCADA

Entre las compañías de renombre mundial que ofrecen entornos de desarrollo orientados al sector industrial, para el desarrollo de sistemas SCADA, podemos citar a los siguientes:

- LabVIEW –National Instrument
- Intouch-Wonderware
- WinCC –Siemens
- Fix

Estas compañías ofrecen soluciones integrales altamente productivas, para la construcción de sistemas de adquisición y visualización de datos, instrumentación y control; Al darnos la capacidad de crear rápidamente una interfaz de usuario que nos proporciona la interactividad con estos sistemas.

Están concebidos para la visualización y manejo de procesos, líneas de fabricación, máquinas e instalaciones. El volumen de funciones de estos paquetes incluye además la emisión de avisos de eventos en una forma adecuada para la aplicación industrial.

Estas herramientas hacen uso de la programación G (Programación Grafica), que difiere de otros lenguajes de programación como C++, Java C#, etc., en que éstos están basados en texto, mientras que G es una programación gráfica.

Sus características más importantes se pueden resumir en:

- Arquitectura de desarrollo abierta (programación en C).
- Soporte de tecnologías Active X.
- Comunicación con otras aplicaciones vía OPC.
- Comunicación sencilla mediante drivers (código que implementa el protocolo de comunicaciones con un determinado equipo inteligente) implementados.
- Programación online: no es necesaria detener la runtime del desarrollo para poder actualizar las modificaciones en la misma.

1.4 OBJETIVOS

1.4.1 OBJETIVO GENERAL

Diseñar e implementar componentes de software para el desarrollo de aplicaciones SCADA, que permitan reducir tiempos de desarrollo de sistemas de información orientados al sector industrial.

1.4.2 OBJETIVOS ESPECÍFICOS

- Recopilar información bibliográfica de los principales conceptos que intervienen en la construcción de nuestro prototipo.
- Determinar las variables que participan con mayor frecuencia en los procesos industriales.
- Diseñar e implementar componentes que interactúen con variables industriales que representan algún proceso productivo.
- Realizar pruebas con datos generados por software, por cada control que se construya para verificar su validez y funcionamiento, elaborar ajustes y correcciones necesarios en caso contrario.
- Realizar diferentes pruebas de funcionamiento del prototipo en conjunto, a través de un enlace alámbrico hacia la computadora, examinar dichos resultados y proporcionar recomendaciones que sirvan en trabajos afines futuros.

1.5 JUSTIFICACIÓN

La realización de este proyecto es importante porque permite:

- Brindar a los desarrolladores de sistemas SCADA, controles de usuario, que reduzcan el tiempo de implementación, evitando tener que construir uno, para cada necesidad.
- Comunicar ámbitos cuya interacción no es muy frecuente (Hardware y Software).
- Orientar el desarrollo de aplicaciones al rubro industrial, poco abordado, evitando así la sobrecarga y bajos precios del software de gestión.

- Utilizar la capacidad del talento nacional existente, dejando de lado una realidad problemática social de conformismo al ser meros importadores y consumidores del conocimiento que otros generan; participando y protagonizando cuanto sea posible de la innovación tecnológica de éste y muchos otros sectores de la ciencia.

- Reducir los costos a niveles alcanzables por parte de las empresas interesadas, puesto que las herramientas existentes para desarrollar sistemas SCADA, como: LabView, Intouch o WinCC, limitan el acceso a la tecnología, a empresas medianas y pequeñas, por tener un costo elevado en sus licencias de uso.

1.6 LIMITACIONES

Las limitaciones dentro de las cuales este proyecto se llevó a cabo comprenden:

- Para la parte de pruebas, no se cuenta con el apoyo de una industria, de la cual obtener datos reales de alguno de sus procesos productivos. Para soslayar este punto, se hará uso de microcontroladores, que interactuaran con sensores de temperatura y presión recogiendo los datos que éstos generan, para luego enviarlos al puerto serie mediante una comunicación directa con la computadora.

- No se cuenta con bibliografía sobre el tema, ni reglas sobre las mejores técnicas para el desarrollo de este tipo de controles, así que la lógica presentada en esta investigación es fruto de nuestra propia experiencia y de algunos tips hallados en foros especializados, por tanto no son la última palabra ni tampoco por ello dejan de ser válidos.

1.7 LÍMITES

El presente proyecto cuenta con varios elementos e indicadores, las cuales no se tomarán en cuenta; se mencionan a continuación los siguientes:

- Quedarán excluidos la implementación de controles para el almacenamiento de variables históricas, la muestra de estadísticas y reportes, ello considerando la limitación de tiempo.

- No se creara una funcionalidad para agregar los controles de forma automática a la barra de herramientas del Microsoft Visual Studio.

- Para la parte de pruebas se usaran sensores de temperatura y presión y se enviaran los datos de éstos a través del puerto serie mediante el uso de microcontroladores, usando una comunicación directa hacia la computadora, no se implementaran mecanismos para obtener datos mediante Ethernet ni otras modalidades.

- Los controles desarrollados se ejecutaran en un sistema operativo Windows que soporte el Framework 3.5 o superior, por tanto no serán válidos en otras arquitecturas o entornos de desarrollo.

1.8 ALCANCES

Los alcances dentro de las cuales este proyecto se llevara a cabo comprenden:

- La investigación abarca el desarrollo de componentes básicos relacionados a variables que participan con mayor frecuencia en los sistemas SCADA.

- Las pruebas en conjunto de este trabajo, se realizaran usando sensores de temperatura y presión, que serán recogidos por un Microcontrolador, por ser muy económicos y sencillos de programar; pues todos aceptan para su programación el estándar ANSI C mientras que los PLCs se rigen al software de cada fabricante.

- Para la fase de pruebas de este proyecto se necesitara un equipo con las características descritas en los siguientes cuadros, las mismas que coinciden con las características mínimas requeridas por el Net Framework 3.5 SP1 , versión inicial en la que se incluye WPF:

Sistema operativo	Versión 4	Versión 3.5
Windows 7 Ultimate x86	√	√
Windows 7 Ultimate N	√	√
Windows 7 Ultimate x64	√	√
Windows 7 Enterprise x86	√	√
Windows 7 Enterprise N	√	√
Windows 7 Enterprise x64	√	√
Windows 7 Professional x86	√	√
Windows 7 Professional N	√	√
Windows 7 Professional x64	√	√
Windows 7 Home Premium x86	√	√
Windows 7 Home Premium N	√	√
Windows 7 Home Premium x64	√	√
Windows7 Home Basic x86	√	√
Windows 7 Home Basic N	√	√
Windows 7 Starter x86	√	√
Windows 7 Starter N	√	√
Windows Vista Ultimate	√	√

Windows Vista Ultimate x64 Edition	√	√
Windows Vista Enterprise	√	√
Windows Vista Enterprise x64	√	√
Windows Vista Business	√	√
Windows Vista Business x64 Edition	√	√
Windows Vista Home Premium	√	√
Windows Vista Home Premium x64 Edition	√	√
Windows Vista Home Basic	√	√
Windows Vista Starter	√	√
Windows XP Professional	√	√
Windows XP Professional x64 Edition	√	√
Windows XP Home Edition	√	√

Tabla 1: Versiones de Windows que soportan WPF
Fuente: http://msdn.microsoft.com/en-us/library/vstudio/w0x726c2%28v=vs.100%29.aspx
Acceso: 10 de Diciembre del 2012

1.9 METODOLOGÍA

La metodología de desarrollo de software usada para el presente proyecto es el modelo del Ciclo de Vida en Cascada o Clásica. Este método modela el ciclo convencional de la Ingeniería del Software, aplicando un enfoque sistemático y secuencial de desarrollo que comienza con la ingeniería del sistema y progresa a través de la captura de requisitos, análisis, diseño, codificación y pruebas.

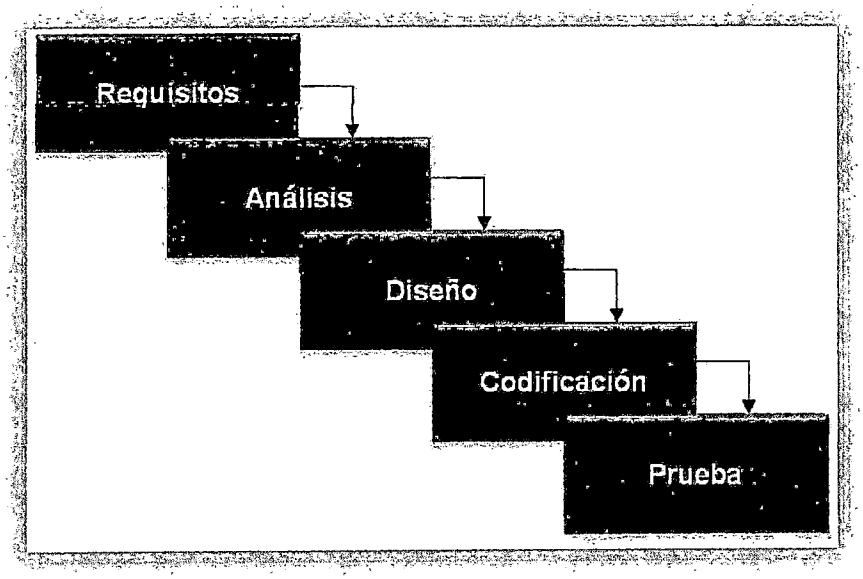


Figura 3: Ciclo de Vida en el Desarrollo de Software en Cascada.
Fuente: http://softwarepalooza.blogspot.com/2012/02/desarrollo-en-cascada.html.
Acceso: 6 de Mayo del 2013.

Se eligió esta metodología por ser la más adecuada al proyecto presentado, además que es muy útil puesto que ayuda a los desarrolladores a comprender qué es lo que se tiene que realizar en cada momento.

Para la fase de Análisis se hará uso de la técnica "Tarjetas CRC" (Clase, Responsabilidad y Colaboración) una técnica que nos permite ver las clases de un sistema, como algo más que un repositorio de datos, sino conocer el comportamiento de cada una en un alto nivel. Es decir las tarjetas CRC permiten documentar una clase de manera formal.

El modelado de clases, responsabilidades y colaboraciones (CRC) aporta un medio sencillo de identificar y organizar las clases que resulten relevantes al sistema o requisitos del producto. Las responsabilidades son los atributos y operaciones relevantes para la clase, o sea, una responsabilidad es "cualquier cosa que conoce o hace la clase". Los colaboradores son aquellas clases necesarias para proveer a otra clase con la información necesaria para completar una responsabilidad.

Cabe indicar también que un modelo CRC es realmente una colección de tarjetas que representan clases. Las tarjetas están divididas en tres secciones. A lo largo de la cabecera de la tarjeta se escribe el nombre de la clase. En el cuerpo se listan las responsabilidades de la clase; a la izquierda y a la derecha los colaboradores.

1.10 PLAN DE EJECUCIÓN

1.10.1 ACTIVIDADES

Id	*Actividades*	*Inicio*	*Fin*
1	Investigación teórica sobre sistemas SCADA y creación de controles de usuario	02/07/12	31/08/12
2	Captura de Requisitos	01/09/12	30/09/12
3	Análisis	01/10/12	31/10/12
3	Desarrollo del Proyecto	01/11/12	31/03/13
	Pruebas	01/04/13	30/04/13
4	Conclusiones y Cierre del proyecto	01/05/13	31/05/13

Tabla 2: Actividades a realizar para el desarrollo de proyecto.

Fuente: Fuente propia.

1.10.2 DIAGRAMA DE GANTT

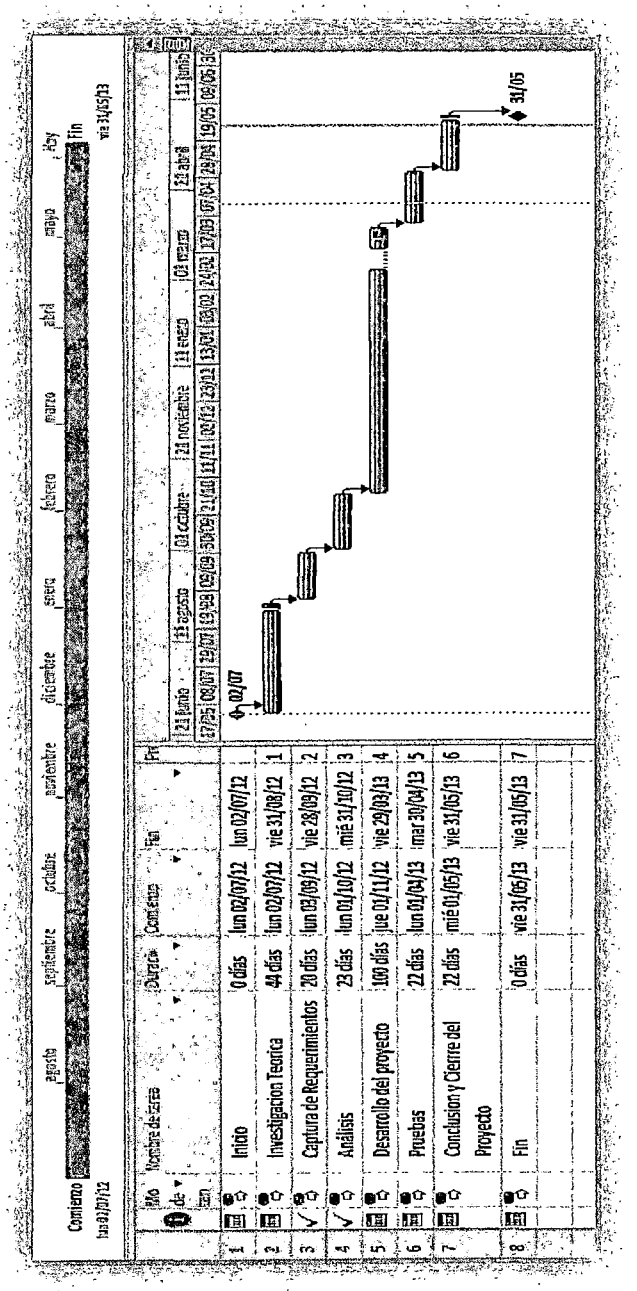


Figura 4: Cronograma del proyecto.
Fuente: Fuente propia.

CAPÍTULO 2

REVISIÓN BIBLIOGRÁFICA

En el presente capítulo se dará a conocer la definición de los principales conceptos, en los que este proyecto se lleva a cabo.

2.1 RESEÑA HISTÓRICA DE LOS SISTEMAS DE VISUALIZACIÓN INDUSTRIAL[1]

De una forma u otra cada vez que se realiza el control de un sistema grande o pequeño, es necesario tener información visual de cómo está funcionando.

Así a medida que los sistemas de control han ido evolucionando y se han hecho cada vez más complejos ha aumentado también la complejidad de los elementos que proporcionan la información al usuario. De un simple indicador de aguja que representa una variable del proceso (por ejemplo la presión de aire en una instalación neumática), se ha llegado a grandes paneles sinópticos que muestran el estado de grandes instalaciones por ejemplo una refinería.

Si nos ceñimos a la era moderna las necesidades de ver en la distancia y controlar una maquina aparecen en los primeros cuadros de control, donde multitud de luces indicaban las diferentes situaciones previstas de la máquina. Cualquier situación imprevista o pasada por alto, podía significar varias horas de trabajo del electricista para llevar la señal olvidada al panel de control y podía ser que no hubiera espacio para colocar el indicador.

La aparición de la informática permitió realizar este tipo de control de manera más sencilla. Así los grandes cuadros de control empezaban a convertirse en monitores que podían mostrar la misma información, y cualquier cambio en la presentación era más sencillo de realizar, bastaban unas modificaciones en el código de la aplicación para que en la pantalla apareciera por ejemplo un contador de piezas olvidado.

Vista la necesidad, varios fabricantes desarrollaron entonces paquetes de software capaces de comunicarse con los sistemas de control existentes y permitieron así una flexibilidad de uso no imaginada hasta el momento, esta tendencia ha ido en aumento de tal manera que hoy en día las opciones existentes son numerosísimas, algunos de los más conocidos:

Intellution	IFIX
Siemens	WinCC
Wonderware	InTouch
National Instrument	LabView
GE-Fanuc	Cimplicity

Tabla 3 : Empresas que ofrecen paquetes para el desarrollo de sistemas SCADA
Fuente: Sistemas SCADA, Aquilino Penin

[1]Bibliografía - Textos [20]

La evolución de los sistemas operativos ha incrementado también las posibilidades de estos sistemas, permitiendo los sistemas distribuidos, gracias a los sistemas de red informáticos.

Con la irrupción de internet en el mundo de las comunicaciones industriales, ahora es posible conectarse con un sistema de control situado en cualquier parte del mundo que gracias a la tecnología Web-Server, un ordenador dotado de un navegador y la dirección IP del sistema que queremos visualizar serían suficientes.

A lo explicado anteriormente se le une, de forma inevitable, la forma en la cual las señales se intercambian entre el sistema a controlar y el sistema que controla. Aparece el concepto de telemetría (medida a distancia), entendido como la tecnología que permite la medición remota de magnitudes físicas y el posterior envío (mediante comunicación inalámbrica, teléfono, redes de ordenadores, enlace de fibra óptica, etcétera) de la información hacia el operador del sistema.

Si además la presentación de los datos se realiza de forma entendible, ya nos proporciona la base para el desarrollo de un sistema de control y monitorización a distancia.

2.2 MARCO CONCEPTUAL

2.2.1 CONCEPTOS Y TERMINOS ELECTRÓNICOS

2.2.1.1 SISTEMAS DE CONTROL[2]

Un sistema de control está definido como un conjunto de componentes que pueden regular su propia conducta o la de otro sistema con el fin de lograr un funcionamiento predeterminado, de modo que se reduzcan las probabilidades de fallos y se obtengan los resultados buscados. Los sistemas de control, se aplican en esencia para los organismos vivos, las máquinas y las organizaciones.

El sistema recibe unas acciones externas o variables de entrada, y cuya respuesta a estas acciones externas son las denominadas variables de salida. Las acciones externas al sistema se dividen en dos grupos, variables de control, que se pueden manipular, y perturbaciones sobre las que no es posible ningún tipo de control.

[2] Bibliografía - Direcciones Electrónicas [3]

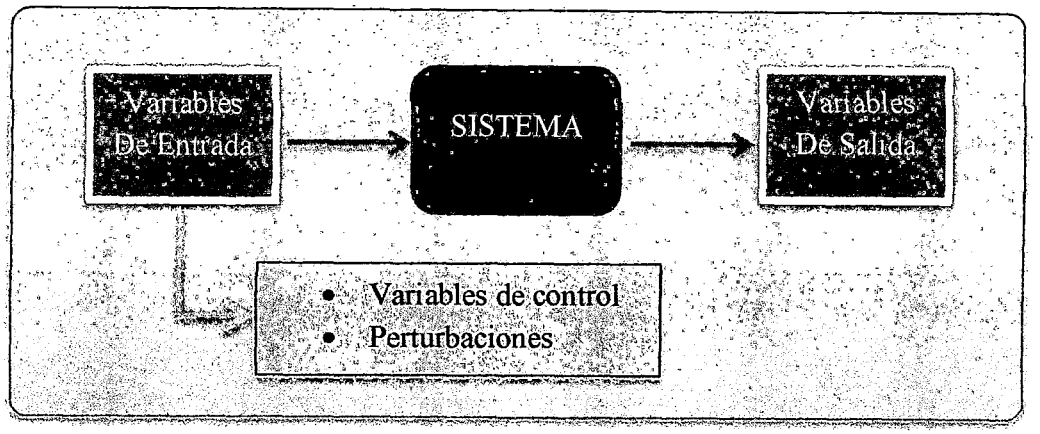


Figura 5 . Esquema general de un sistema.
Fuente: Fuente Propia.

La finalidad de un sistema de control es conseguir, mediante la manipulación de las variables de control, un dominio sobre las variables de salida, de modo que estas alcancen unos valores prefijados (consigna). En otras palabras la finalidad es controlar el funcionamiento de una máquina o de un proceso.

Hoy en día los sistemas de control son síntomas del proceso industrial que estamos viviendo. Estos sistemas se usan típicamente en sustituir un trabajador pasivo que controla un determinado sistema (ya sea eléctrico, mecánico, etc.) con una posibilidad nula o casi nula de error y un grado de eficiencia mucho más grande que el de un trabajador.

SCADA y HMI son los sistemas de control que se utilizan en cualquier organización.

2.2.1.2 INTERFAZ HOMBRE MAQUINA

HMI son las siglas de Human Machine Interface, Interfaz Hombre Maquina en español. Es el aparato que presenta los datos a un operador (humano) y a través del cual éste controla el proceso. Los sistemas HMI podemos pensarlos como una "ventana" de un proceso. Esta ventana puede estar en dispositivos especiales como paneles de control o en una computadora. Los sistemas HMI en computadoras se los conoce también como software HMI o de monitoreo y control de supervisión.

Cuando los seres humanos y las maquinas interactúan lo hacen a través de un medio o interfaz, que definimos como HMI, por tanto es cualquiera que sea la manera de presentar datos ante un operador y/o permitirle acceder al control de un sistema

La HMI es el punto en el que seres humanos y maquinas se ponen en contacto, transmitiéndose mutuamente tanto información, órdenes y datos como sensaciones,

intuiciones y nuevas formas de ver las cosas. Por otro lado, la interfaz es también un límite a la comunicación en muchos casos, ya que aquello que no sea posible expresar a través de ella permanecerá fuera de nuestra relación mutua. Es así como en muchos casos la interfaz se convierte en una barrera debido a un pobre diseño y una escasa atención a los detalles de la tarea a realizar. Si la interfaz está bien diseñada, el usuario encontrará la respuesta que espera a su acción; si no es así, puede ser frustrante para el usuario, que habitualmente tiende a culparse a sí mismo por no saber usar el objeto.

Las señales de los procesos son conducidas al HMI por medio de dispositivos como tarjetas de entrada/salida en la computadora, Microcontroladores, PLC's, RTU, etc.

Figura 6: HMI para una tele operación inmersiva de un Robot.
Fuente: http://www.hizook.com/blog/2009/08/17/immersive-man-machine-interface-teleoperation-rollin
Acceso: 10 de Diciembre del 2012.

2.2.1.3 SISTEMAS SCADA[3]

SCADA Proviene de las siglas "Supervisory Control And Data Adquisition" (Supervisión, Control y Adquisición de Datos). Permite controlar y supervisar instalaciones de cualquier tipo y /o procesos industriales de características variadas a distancia.

I. ***Supervisión;*** (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. La evolución de las variables de control, se observan desde un monitor.

II. ***Control;*** Para modificar la evolución del proceso, actuando sobre los reguladores autónomos básicos (consignas, alarmas, menús, etc.) o directamente sobre el proceso mediante las salidas conectadas.

III. ***Adquisición de datos;*** Para recoger y procesar la información recibida. Se refiere al método usado para acceder y controlar información o datos de los equipos que se están controlando y supervisando. Los datos recogidos son luego remitidos a un sistema de telemetría listo para ser transferidos a los diferentes sitios.

Los sistemas SCADA facilitan la comunicación en tiempo real con los dispositivos de campo (controladores autónomos, autómatas programables, sensores, actuadores, registradores, etc.), controlando el proceso automáticamente desde la pantalla del ordenador, por medio de un software especializado. Proveen de toda la información que se genera en el proceso productivo (supervisión, control de calidad, control de producción, almacenamiento de datos, etc.) y permite su gestión e intervención. Además, envía la información generada en el proceso productivo a diversos usuarios, tanto del mismo nivel como hacia otros supervisores dentro de la misma empresa, es decir, que permite la participación de otras áreas como por ejemplo: control de calidad, supervisión, mantenimiento, etc.

El sistema SCADA usualmente presenta la información al personal operativo de manera gráfica, en forma de un diagrama de representación. Esto significa que el operador puede ver un esquema que representa la planta que está siendo controlada. Todo este proceso se ejecuta normalmente en tiempo real, y están diseñados para dar al operador de planta la posibilidad de supervisar, controlar dichos procesos y tomar decisiones en base a los resultados que visualiza en la pantalla.

[3] Bibliografía – Textos [20]

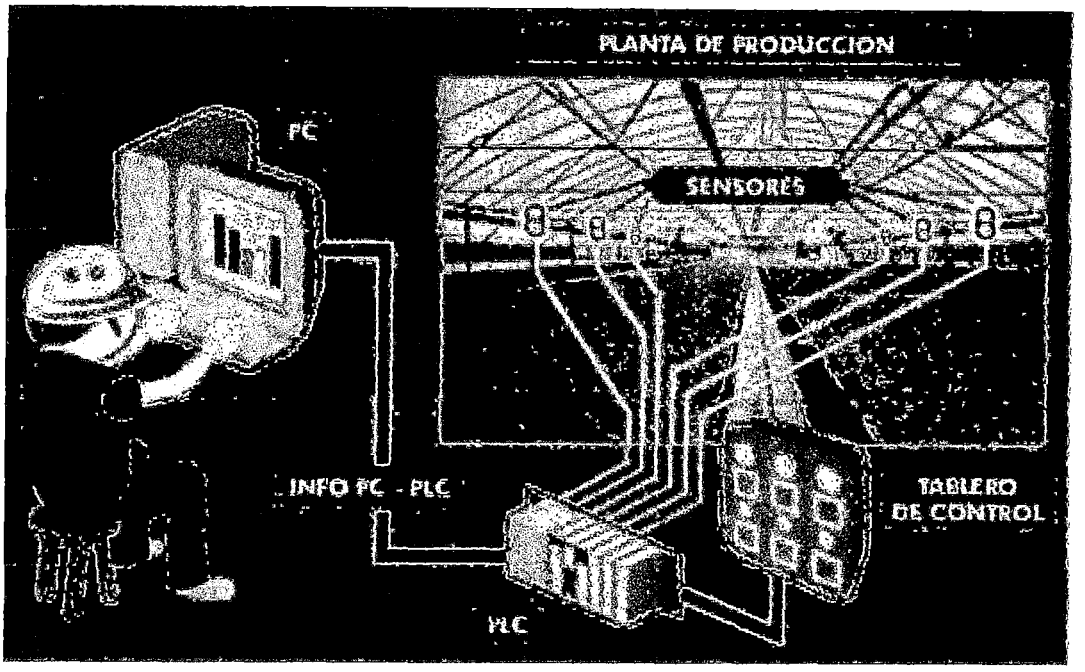


Figura 7: Ejemplo de un sistema SCADA.
Fuente: http://serviciotecnicodelsureste.blogspot.com/2010/07/blog-post.html.
Acceso el 22 de marzo del 2013.

2.2.1.3.1 FUNCIONES DE UN SISTEMA SCADA

Las funciones son las tareas que le están permitidas realizar al operador, tareas con las cuales debe de cumplir el sistema SCADA y entre las cuales tenemos las siguientes:

I. ***Control remoto de instalaciones y equipos***; Con ayuda de este sistema se puede encender o apagar cualquier equipo de manera remota, automática y manual como por ejemplo encender motores, abrir válvulas, activar alguna máquina, etc., permitiéndonos ajustar valores de referencia, parámetros, etc.

II. ***Supervisión de instalaciones y equipos***; El operador es capaz de conocer el estado en que se encuentran las instalaciones y los distintos equipos que son controlados por el sistema, lo que le permitirá dirigir las tareas de mantenimiento, estadísticas de fallas, entre otras, si así lo requiriera el equipo.

III. ***Procesamiento de datos***; La información que es recabada del proceso es analizada y comparada con datos anteriores y con datos de otros puntos de

referencia obteniendo como resultado una información más confiable y verídica.

IV. ***Generación de reportes***; El sistema debe de permitir generar reportes con datos estadísticos del proceso en un lapso de tiempo o en tiempo real, el cual es determinado por el operador y las necesidades del proceso.

V. ***Visualización gráfica dinámica***; Un sistema SCADA es capaz de darnos imágenes en movimiento, con ayuda de estas imágenes se podrá representar el comportamiento del proceso, dándole al operador la impresión de estar dentro del lugar en donde se está llevando a cabo el proceso si éste se encontrase en un lugar lejano del mismo.

VI. ***Almacenamiento de información histórica***; Nos permite almacenar los datos adquiridos del proceso para poder ser analizada posteriormente y el tiempo en que permanecerá almacenada dependerá del operador o de aquella persona que se encargó de realizar el programa para la supervisión y control del proceso.

VII. ***Representación de señales de alarma;*** Gracias a esta función se puede alertar al operador de cualquier falla o condición que se encuentre fuera de los parámetros preestablecidos que pudieran surgir en el equipo durante el proceso; éstas señales pueden ser tanto sonoras como visuales.

VIII. ***Programación de eventos***; Nos brinda la posibilidad de programar subprogramas que nos brinden de manera automática reportes, gráficas de curvas, estadísticas, entre otras tareas.

2.2.1.3.2 ARQUITECTURA DE UN SISTEMA SCADA[4]

Cuando hablamos de arquitectura nos estamos refiriendo a los elementos que conforman un sistema SCADA; siendo estos de suma importancia, ya que de ellos dependerá en gran medida el buen funcionamiento no solo del mismo sino también del proceso al cual se le desee incorporar este sistema. Un sistema SCADA hecho y derecho se compone de muchísimas partes, las más principales son las siguientes:

I. ***HMI;*** Esto se utiliza para conectar con todos los procesos y luego presentar estos datos a un operador humano. El operador hace uso de todos los datos y por lo tanto monitoriza y controla todos los procesos. El HMI es crítico para el éxito de cualquier SCADA, en como este aparato proporciona todos los datos a

[4] Bibliografía – Textos [22]

un operador humano. El operador normalmente recibe información de HMI en forma de gráficos o diagramas. Esta entrada es analizada por el operador para tomar decisiones en los procesos en consecuencia.

II. ***Instrumentación de Campo***; La instrumentación de campo se refiere a los sensores y actuadores que están directamente comunicados a la planta o equipo. Generan las señales analógicas y digitales que serán supervisadas por la Estación Remota. En la instrumentación de campo la prioridad absoluta es obtener la máxima precisión y la más alta fiabilidad, en los procesos industriales no hay casi nada más importante, que medir, posicionar, registrar y regular.

III. ***Estaciones Remotas***; La Estación Remota se instala en una planta remota o equipos que se están supervisando por la computadora central. Esto puede ser una Unidad Terminal Remota (RTU) o un Controlador Lógico Programable (PLC).

IV. ***PLC***; Estos son los controladores lógicos programables normalmente utilizados como dispositivos de campo. Estos son dispositivos flexibles y de bajo costo.

V. ***RTU***; Estas son unidades terminales remotas a las que se conectan sensores utilizados en los procesos. Convierten señales en datos digitales y las envían al sistema de supervisión.

VI. ***Red de Comunicación***; La red de comunicación se refiere al equipo de comunicación necesario para transferir datos a y de diferentes sitios. El medio usado puede ser cable, teléfono o radio. El uso del cable usualmente se implementa en factorías. Esto no es práctico para sistemas que cubren áreas extensas por el alto coste de los cables, conducciones y el trabajo de instalarlos. El uso de líneas de teléfono es una solución más barata para sistemas con cobertura grande. Las líneas alquiladas se usan para sistemas que requieren conexión on-line con estaciones remotas. En los sitios remotos normalmente no hay líneas telefónicas accesibles. En estos ambientes el uso de radio ofrece una solución económica. Los módems de radio se usan para conectar los sitios remotos al anfitrión. Una operación on-line también puede ser implementada en el sistema de radio. Para localizaciones donde un link de radio directo no puede ser establecido, se usa un repetidor de radio para conectar estos sitios.

VII. ***Estación Maestra o Estación Central de Supervisión***; Se refiere a los servidores y al software responsable para comunicarse con el equipo del campo (RTUs, PLCs, etc) en éstos se encuentra el software HMI corriendo para las estaciones de trabajo en el cuarto de control, o en cualquier otro lado. En un sistema SCADA pequeño, la estación maestra puede estar en un solo computador, A gran escala en los sistemas SCADA la estación maestra puede incluir muchos servidores, aplicaciones de software distribuido, y sitios de recuperación de desastres.

La Estación de Supervisión Central (Central Monitoring Station CMS) es la unidad maestra del sistema SCADA. La CMS puede tener una configuración de computadora simple o puede estar conectada en red para permitir compartir información desde el sistema SCADA. El programa de interfaz hombre máquina estará funcionando en el computador CMS. Un diagrama de toda la planta se mostrará en la pantalla para una identificación más fácil del sistema real.

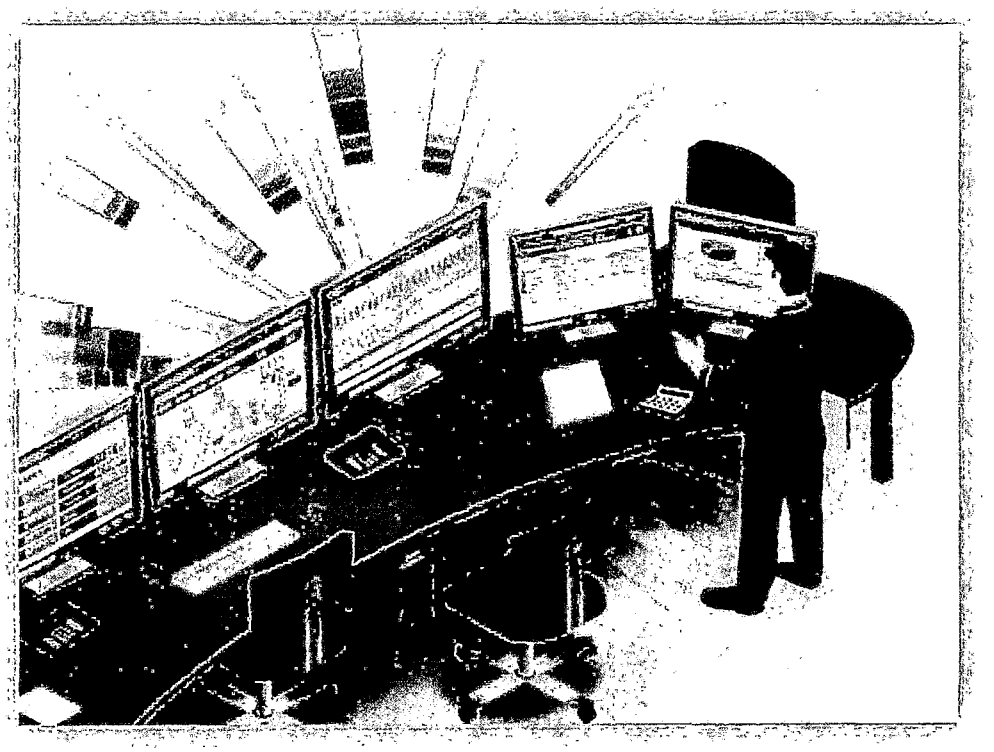


Figura 8: Estación maestra o sala de control de un sistema SCADA.
Fuente: http://www.copadata.com/es/products/zenon-supervisor.html.
Acceso: 27 de febrero de 2013.

VIII. ***Operario;*** En la función de dialogo hombre-máquina, el operador desempeña un papel importante. En base a los datos de los que dispone, debe realizar acciones que condicionan el buen funcionamiento de máquinas e instalaciones sin comprometer seguridad ni disponibilidad. Es por tanto indispensable que la calidad del diseño de las interfaces y de la función de dialogo garantice al operador la posibilidad de actuar con seguridad en todo momento.

2.2.1.3.3 CAMPO DE ACCIÓN DE LOS SISTEMAS SCADA[5]

Los sistemas SCADA proporcionan soluciones que permiten contar con automatización de procesos complejos en donde las situaciones sean imprácticas para el control humano, como plataformas petroleras, refinerías, ductos, pozos petroleros, transformadores eléctricos, líneas de transmisión de alta tensión, medidores de agua, gas, eléctricos, monitoreo remoto de sitios sin personal, sistemas con factores de control complejos y de movimiento rápido.

SCADA tiene aplicaciones en la fabricación, producción, generación de energía, refinación y muchos otros sectores de la economía, incluso los procesos en instalaciones como aeropuertos, estaciones de ferrocarril, barcos y estaciones espaciales hacen uso de SCADA para monitorear y controlar los distintos procesos.

2.2.1.4 PROCESOS INDUSTRIALES

Se entiende por proceso a las diferentes etapas que componen de una manera ordenada y escalonada, la realización de alguna cosa. Es todo desarrollo sistemático que conlleva una serie de pasos ordenados, los cuales se encuentran estrechamente relacionados entre sí y cuyo propósito es llegar a un resultado preciso; de forma general el desarrollo de un proceso conlleva una evolución en el estado del elemento sobre el que se está aplicando dicho tratamiento hasta que este desarrollo llega a su fin.

Un proceso puede ser descrito como la secuencia de cambios en una sustancia. Ésta secuencia de cambios puede ocurrir en el aspecto químico, físico o ambos, en la composición de una sustancia incluyendo parámetros como el flujo, nivel, presión, temperatura densidad volumen, acidez y gravedad específica, así como muchos otros, también muchos procesos requieren de transferencia de energía. La mezcla de fluidos, el calentamiento o el enfriamiento de substancias, el bombeo de agua de un lugar a otro, el enlatado de comida, la destilación de gasolina, el pasteurizado de la leche, y convertir la luz solar en energía eléctrica todos pueden ser descritos como procesos.

Los procesos industriales tienen como propósito principal el de transformar materias primas en un producto final. Durante el proceso de la producción de estos bienes, se tienen diversos procesos, ya sea que sean reutilizados los materiales, o se convierta energía para producir el producto final.

[5] Bibliografía – Textos [21]

Un proceso industrial comienza con la medición de una variable. Por ejemplo, la temperatura del fluido del proceso fuera del intercambiador de calor es medida. Esta información es utilizada para llevar a cabo una decisión acerca el proceso. Finalmente, se lleva a cabo la acción basada en la decisión tomada.

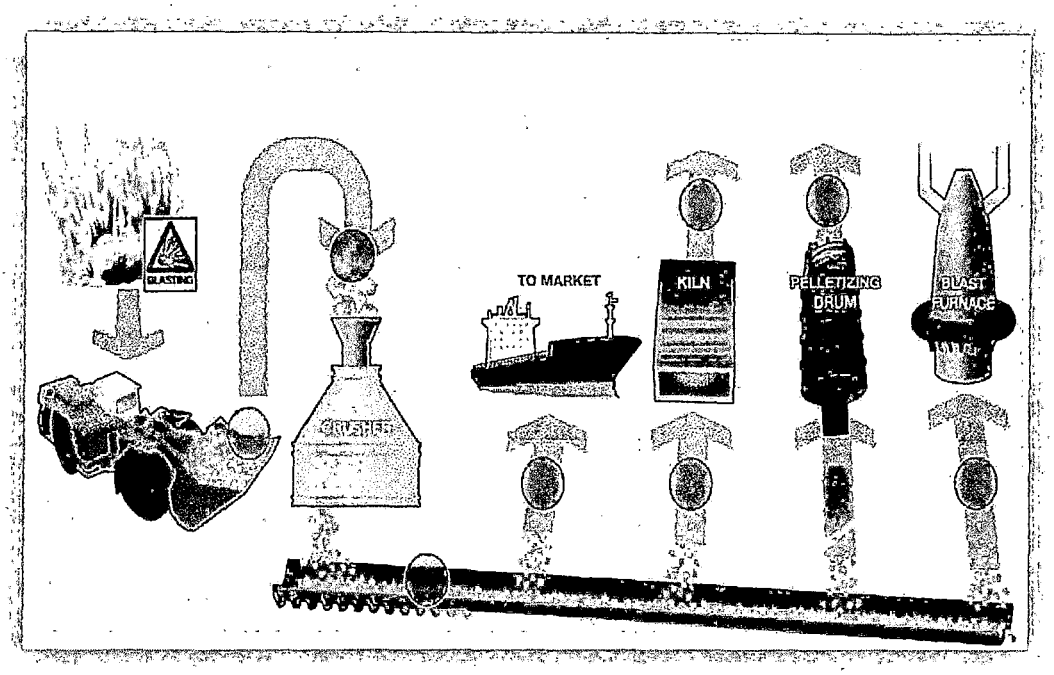


Figura 9: Proceso Productivo de Materiales y Retroalimentación a Diversos Procesos Industriales. Fuente: http://www.ltu.se/ltu/media/news/Forskare-gor-nyttiga-datorprogram-till-industrin-1.97809?l=en Acceso: 06 de marzo de 2013.

2.2.1.5 INSTRUMENTACIÓN DE CAMPO

La instrumentación es la tecnología del uso de instrumentos para la medición. Instrumentación es el grupo de elementos que sirven para medir, convertir, transmitir, controlar o registrar variables de un proceso con el fin de optimizar los recursos utilizados en éste. En otras palabras la instrumentación es la ventana a la realidad de lo que está sucediendo en determinado proceso, lo cual servirá para determinar si el mismo va encaminado hacia donde deseamos y de no ser así podremos usar estos para actuar sobre algunos parámetros del sistema y proceder de forma correctiva.

La instrumentación es lo que ha permitido el gran avance tecnológico de la ciencia actual, por ejemplo la automatización de procesos industriales, que solo es posible a través de elementos que puedan censar lo que sucede en el ambiente para luego tomar una acción de control pre-programada que actué sobre el sistema para obtener el resultado previsto.

Los procesos industriales son muy variados y abarcan muchos tipos de productos: la fabricación de los productos derivados del petróleo, de los productos alimenticios, la industria cerámica, las centrales generadoras de energía, la siderurgia, los tratamientos térmicos, la industria papelera, la industria textil, etc. En todos estos procesos es absolutamente necesario controlar y mantener constantes algunas magnitudes, tales como la presión, el caudal, el nivel, la temperatura, el pH, la conductividad, la velocidad, la humedad, el punto de rocío, etcétera. Los instrumentos de medición y control permiten el mantenimiento y la regulación de estas constantes en condiciones más idóneas que las que el propio operador podría realizar.

En los inicios de la era industrial, el operario llevaba a cabo un control manual de estas variables utilizando sólo instrumentos simples, manómetros, termómetros, válvulas manuales, etc., control que era suficiente por la relativa simplicidad de los procesos. Sin embargo, la gradual complejidad con que éstos se han ido desarrollando ha exigido su automatización progresiva por medio de los instrumentos de medición y control. Éstos instrumentos han ido liberando al operario de su función de actuación física directa en la planta y al mismo tiempo, le han permitido una labor única de supervisión y de vigilancia del proceso desde centros de control situados en el propio proceso o bien en salas aisladas separadas; asimismo, gracias a los instrumentos ha sido posible fabricar productos complejos en condiciones estables de calidad y de características, condiciones que al operario le serían imposibles o muy difíciles de conseguir, realizando exclusivamente un control manual.

2.2.1.6 MICROCONTROLADOR[6]

Un micro controlador (abreviado µC, UC o MCU) es un circuito integrado programable, capaz de ejecutar las órdenes grabadas en su memoria. Está compuesto de varios bloques funcionales, los cuales cumplen una tarea específica. Un micro controlador incluye en su interior las tres principales unidades funcionales de una computadora: unidad central de procesamiento, memoria y periféricos de entrada/salida.

Al ser fabricados, la memoria ROM de los microcontroladores no poseen datos. Para que éstos puedan controlar algún proceso es necesario crear y luego grabar en la EEPROM o equivalente del microcontrolador algún programa, el cual puede ser escrito en lenguaje ensamblador u otro lenguaje para microcontroladores; sin embargo, para que el programa pueda ser grabado en la memoria del micro controlador, debe ser codificado en sistema numérico hexadecimal que es finalmente el sistema que hace trabajar al micro controlador cuando éste es alimentado con el

[6] Bibliografía – Direcciones Electrónicas [1]

voltaje adecuado y asociado a dispositivos analógicos y discretos para su funcionamiento.

2.2.1.6.1 CARACTERÍSTICAS

Los microcontroladores son diseñados para reducir el costo económico y el consumo de energía de un sistema en particular. Por eso el tamaño de la unidad central de procesamiento, la cantidad de memoria y los periféricos incluidos dependerán de la aplicación.

Pueden encontrarse en casi cualquier dispositivo electrónico como automóviles, lavadoras, hornos microondas, teléfonos, etc.

Un micro controlador difiere de una unidad central de procesamiento normal, debido a que es más fácil convertirla en una computadora en funcionamiento, con un mínimo de circuitos integrados externos de apoyo. La idea es que el circuito integrado se coloque en el dispositivo, enganchado a la fuente de energía y de información que necesite, y eso es todo.

Un microcontrolador típico tendrá un generador de reloj integrado y una pequeña cantidad de memoria de acceso aleatorio y/o ROM/EPROM/EEPROM/flash, con lo que para hacerlo funcionar todo lo que se necesita son unos pocos programas de control y un cristal de sincronización. Los microcontroladores disponen generalmente también de una gran variedad de dispositivos de entrada/salida, como convertidor analógico digital, temporizadores, UARTs y buses de interfaz serie especializados.

2.2.1.7 CONTROLADOR LÓGICO PROGRAMABLE[7]

Un controlador lógico programable, más conocido por sus siglas en inglés PLC (Programmable Logic Controller), es un equipo electrónico que ha sido diseñado para programar y controlar procesos en tiempo real y es utilizado en la ingeniería automática o automatización industrial, para automatizar procesos electromecánicos, tales como el control de maquinarias.

Se puede pensar en un PLC como un pequeño computador industrial que ha sido altamente especializado para prestar la máxima confianza y máximo rendimiento en un ambiente industrial. En su esencia, un PLC mira sensores digitales y analógicos y switches (entradas), lee su programa de control, hace cálculos matemáticos y como

[7] Bibliografía – Direcciones Electrónicas [2]

resultado controla diferentes tipos de hardware (salidas) tales como válvulas, luces, relés, servomotores, etc. en un marco de tiempo de milisegundos.

Comúnmente los PLCs intercambian información con paquetes de software en el nivel de planta como interfaces maquina operador (HMI) o Control de Supervisión y Adquisición de Datos (SCADA). Todo intercambio de datos con el nivel de negocios de la empresa (servicios de información, programación, sistemas de contabilidad y análisis) tiene que ser recogido, convertido y transmitido a través de un paquete SCADA.

Los PLCs son utilizados en muchas industrias y máquinas. A diferencia de las computadoras de propósito general, el PLC está diseñado para múltiples señales de entrada y de salida, rangos de temperatura ampliados, inmunidad al ruido eléctrico y resistencia a la vibración y al impacto. Hoy en día son los cerebros de la inmensa mayoría de la automatización, procesos y máquinas especiales en la industria.

2.2.1.7.1 CAMPOS DE APLICACIÓN

El PLC por sus especiales características de diseño tiene un campo de aplicación muy extenso. La constante evolución del hardware y software amplía constantemente este campo para poder satisfacer las necesidades que se detectan en el espectro de sus posibilidades reales.

Su utilización se da fundamentalmente en aquellas instalaciones en donde es necesario un proceso de maniobra, control, señalización, etc., por tanto, su aplicación abarca desde procesos de fabricación industriales de cualquier tipo a transformaciones industriales, control de instalaciones, etc.

Sus reducidas dimensiones, la extremada facilidad de su montaje, la posibilidad de almacenar los programas para su posterior y rápida utilización, la modificación o alteración de los mismos, etc., hace que su eficacia se aprecie fundamentalmente en procesos en que se producen necesidades tales como:

- Espacio reducido
- Procesos de producción periódicamente cambiantes
- Procesos secuenciales
- Maquinaria de procesos variables
- Instalaciones de procesos complejos y amplios
- Chequeo de programación centralizada de las partes del proceso

2.2.1.7.2 VENTAJAS

Dentro de las ventajas que estos equipos poseen se encuentra que, gracias a ellos, es posible ahorrar tiempo en la elaboración de proyectos, pudiendo realizar modificaciones sin costos adicionales. Por otra parte, son de tamaño reducido y mantenimiento de bajo costo, además permiten ahorrar dinero en mano de obra y la posibilidad de controlar más de una máquina con el mismo equipo. Sin embargo, y como sucede en todos los casos, los controladores lógicos programables, presentan ciertas desventajas como es la necesidad de contar con técnicos calificados y adiestrados específicamente para ocuparse de su buen funcionamiento.

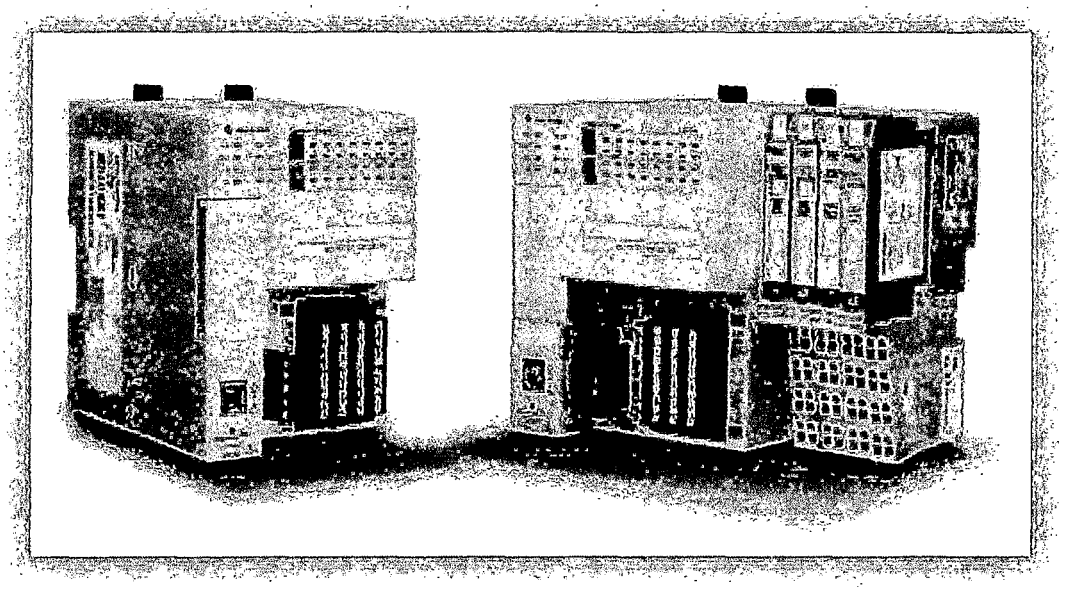


Figura 10: Imagen de un Controlador Lógico Programable
Fuente: http://www.rocatek.com/forum_plc1.php
Acceso: 25 de marzo del 2013

2.2.2 CONCEPTOS Y TERMINOS INFORMÁTICOS

2.2.2.1 PUERTO SERIE

Un puerto serie o puerto serial, también conocido como COM es una interfaz de comunicaciones entre ordenadores y perifericos, el cual envía y recibe información en forma asíncrona, bit por bit enviando un solo bit a la vez, en contraste con el puerto paralelo que envía varios bits simultáneamente. Se denomina "serial" porque el puerto serie "serializa" los datos. Esto quiere decir que toma un byte de datos y transmite los 8 bits del byte de uno en uno.

A lo largo de la mayor parte de la historia de los ordenadores, la transferencia de datos a través de los puertos de serie ha sido generalizada. Se ha usado y sigue usándose para conectar los ordenadores a dispositivos como terminales o módems. Los ratones, teclados, y otros periféricos también se conectaban de esta forma. Actualmente en la mayoría de los periféricos serie, la interfaz USB ha reemplazado al puerto serie por ser más rápida. La mayor parte de los ordenadores están conectados a dispositivos externos a través de USB y, a menudo, ni siquiera llegan a tener un puerto serie.

El puerto serie se elimina para reducir los costes y se considera que es un puerto heredado y obsoleto. Sin embargo, los puertos serie todavía se encuentran en sistemas de automatización industrial y algunos productos industriales y de consumo.Los dispositivos de redes, como los enrutadores y switches, a menudo tienen puertos serie para modificar su configuración. Los puertos serie se usan frecuentemente en estas áreas porque son sencillos, baratos y permiten la interoperabilidad entre dispositivos. La desventaja es que la configuración de las conexiones serie requiere, en la mayoría de los casos, un conocimiento avanzado por parte del usuario y el uso de comandos complejos si la implementación no es adecuada.

El cable para el puerto serie que se destina a comunicar dos equipos por dicho puerto, es un cable denominado "modem nulo". El conector externo para un puerto serial puede ser de 9 o de 25 pines.

El Puerto Serie toma como '1' cualquier voltaje que se encuentre entre –3 y –25 V y como '0', entre +3 y +25 V, a diferencia del Puerto Paralelo, cuyo rango de voltajes esta entre 0 y 5 V.

2.2.2.2 COMPONENTE SERIALPORT DE VISUAL STUDIO

Esta clase proporciona E/S sincrónica y orientada a eventos, acceso a los estados de punto de conexión e interrupción, así como acceso a las propiedades del controlador serie.

En el pasado, antes de la llegada de .Net Framework 2.0, para comunicarse con un puerto serie, había que utilizar la API de Windows o el uso de un control de terceros. Con Net 2.0, Microsoft añadió soporte con la inclusión de la clase SerialPort como parte del espacio de nombres System.IO.Ports. Este nuevo framework provee clases con la capacidad de acceder a los puertos serie de un computador y para comunicarse con dispositivos de E/S seriales. Esta clase usa el estandar RS 232 para la comunicación entre dispositivos.

La clase SerialPort admite las codificaciones siguientes: ASCIIEncoding, UTF8Encoding, UnicodeEncoding, UTF32Encoding y cualquier codificación definida en mscorlib.dll donde la página de códigos sea menor que 50000 o sea 54936.

Se puede usar Serial Port Component para diferentes propósitos:

- Para controlar máquinas industriales a través de un puerto en serie.
- Para configurar dispositivos de red (como por ejemplo, servidores de impresoras, routers) a través de un puerto serie.
- Para controlar un módem conectado a un puerto serie /USB o Bluetooth.
- Para mandar mensajes SMS a un teléfono móvil utilizando SmartPhone GSM / Módem conectado al puerto serie/ USB o Bluetooth de la computadora.
- Para transferir archivos entre computadoras, a través de un cable de módem nulo.
- Para cualquier otro caso en donde haya involucrada comunicación en serie.

En Windows Forms es muy fácil trabajar con éste componente. Es tan sencillo como arrastrar el componente a nuestro Formulario, para luego empezar a trabajar con él.

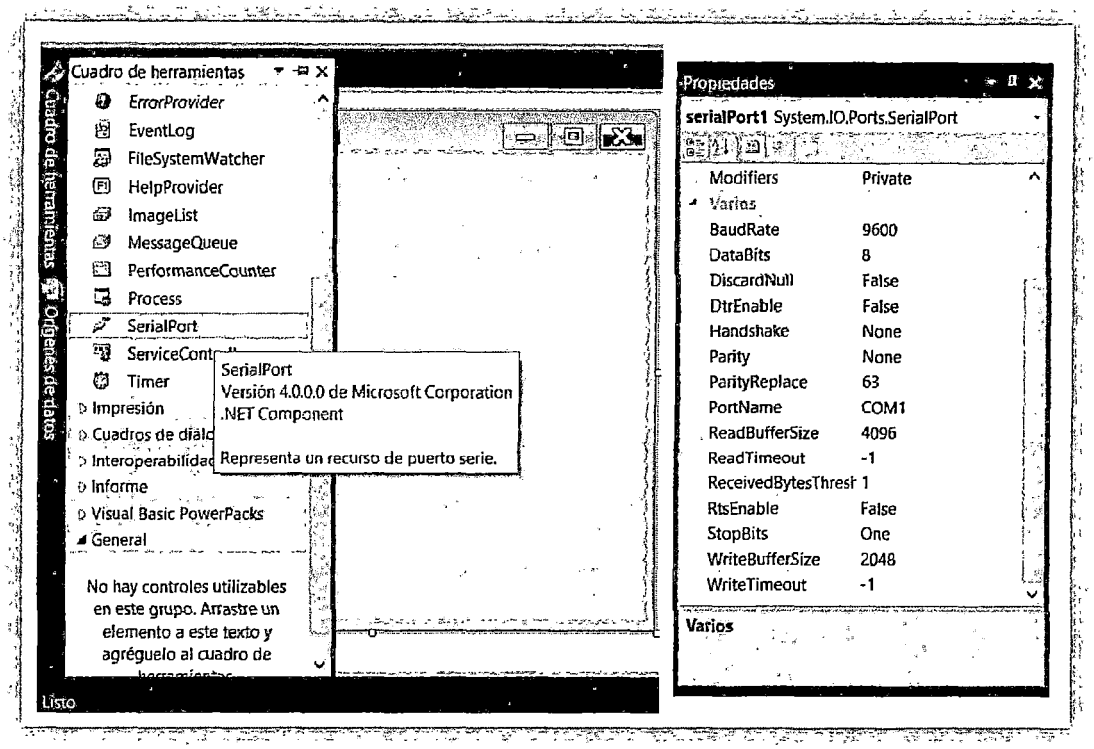


Figura 11: Interacción con el Puerto Serie en Windows Forms.
Fuente: Fuente Propia.

En lo que se refiere a WPF el componente SerialPort no se encuentra disponible en la barra de herramientas, con lo cual el único modo de interactuar con él es mediante programación, el cual nos permite tener un mayor control de éste; la Clase Comunicación se realiza de este modo.

2.2.2.3 COMPONENTES DE SOFTWARE[8]

En el ámbito del software, el término componente se utiliza normalmente para referirse a un objeto reutilizable que expone una o varias interfaces a clientes de forma normalizada. Un componente se puede implementar como clase única o como conjunto de clases; el requisito principal es que la interfaz pública básica esté bien definida. Los componentes proporcionan código reutilizable en forma de objetos. Una aplicación que utiliza el código de un componente para crear objetos y llamar a sus métodos y propiedades se conoce como cliente. Un cliente puede estar o no en el mismo ensamblado que el componente que utiliza.

[8] Bibliografía – Direcciones Electrónicas [14]

En el contexto de .NET Framework, un componente es una clase o un conjunto de clases que implementa la interfaz IComponent o una interfaz derivada directa o indirectamente de una clase que implementa esta interfaz. Un "componente" es un término genérico que significa: "una parte de algo". En realidad no tiene una definición técnica, .NET es un poco más consistente en que por lo general cuando algo se llama un "componente", significa que no tiene una interfaz de usuario. Todo en la ficha Componentes (como Timer) sólo es visible en la parte inferior de la ventana de diseño en Visual Studio (una zona conocida como la "bandeja de componentes").

Un componente es una clase que implementa la interfaz IComponent, que es el requisito para que una clase sea capaz de aparecer en la caja de herramientas de Visual Studio. Generalmente no tomamos IComponent nosotros mismos, sino más bien heredamos de la clase base Component (System.ComponentModel.Component es la implementación de la clase base predeterminada de la interfaz IComponent).

Aunque la mayor parte de los componentes no tiene interfaces visuales, un control es un componente que proporciona una interfaz de usuario, ósea la clase Control sí hereda de la clase Component, como tal, cada control es un componente, por lo que pueden aparecer en el cuadro de herramientas Visual Studio. Algunos de los componentes que más se utilizan en la programación con .NET Framework son los controles visuales que se agregan a los formularios Windows Forms o WPF como Button, ComboBox, etc. Los componentes no visuales incluyen Timer Control, SerialPort y ServiceController, entre otros.

2.2.2.4 CONTROLES DE USUARIO[9]

Los controles de usuario (User Controls) proporcionan un medio para crear y reutilizar interfaces gráficas de usuario. Un control de usuario es esencialmente un componente con una representación visual. Como tal, puede constar de uno o más controles de formularios Windows Forms o WPF, componentes o bloques de código, que pueden extender su funcionalidad mediante la validación de la entrada del usuario, la modificación de las propiedades de presentación o la ejecución de otras tareas requeridas por su autor. Los controles de usuario pueden incluirse en formularios Windows Forms o WPF de la misma manera que los demás controles pues son tratados como una unidad con propiedades y métodos al incluir el código necesario para manipular su contenido e incluso realizar tareas como el enlace de datos.

[9] Bibliografía – Direcciones Electrónicas [11]

Se puede crear controles de usuario si la aplicación requiere funcionalidad que no está disponible en los controles estándar, existentes en el cuadro de herramientas, reutilizando la funcionalidad de éstos.

En ocasiones, es posible que necesitemos cierta funcionalidad en un control que no está incluida en los controles .NET integrados; en estos casos podemos crear nuestros propios controles. Como ejemplo, podemos hacer uso de la propia ventana (que a fin de cuentas es un control más), para redefinirla a nuestra propia necesidad, y darle una apariencia más personalizada, como:

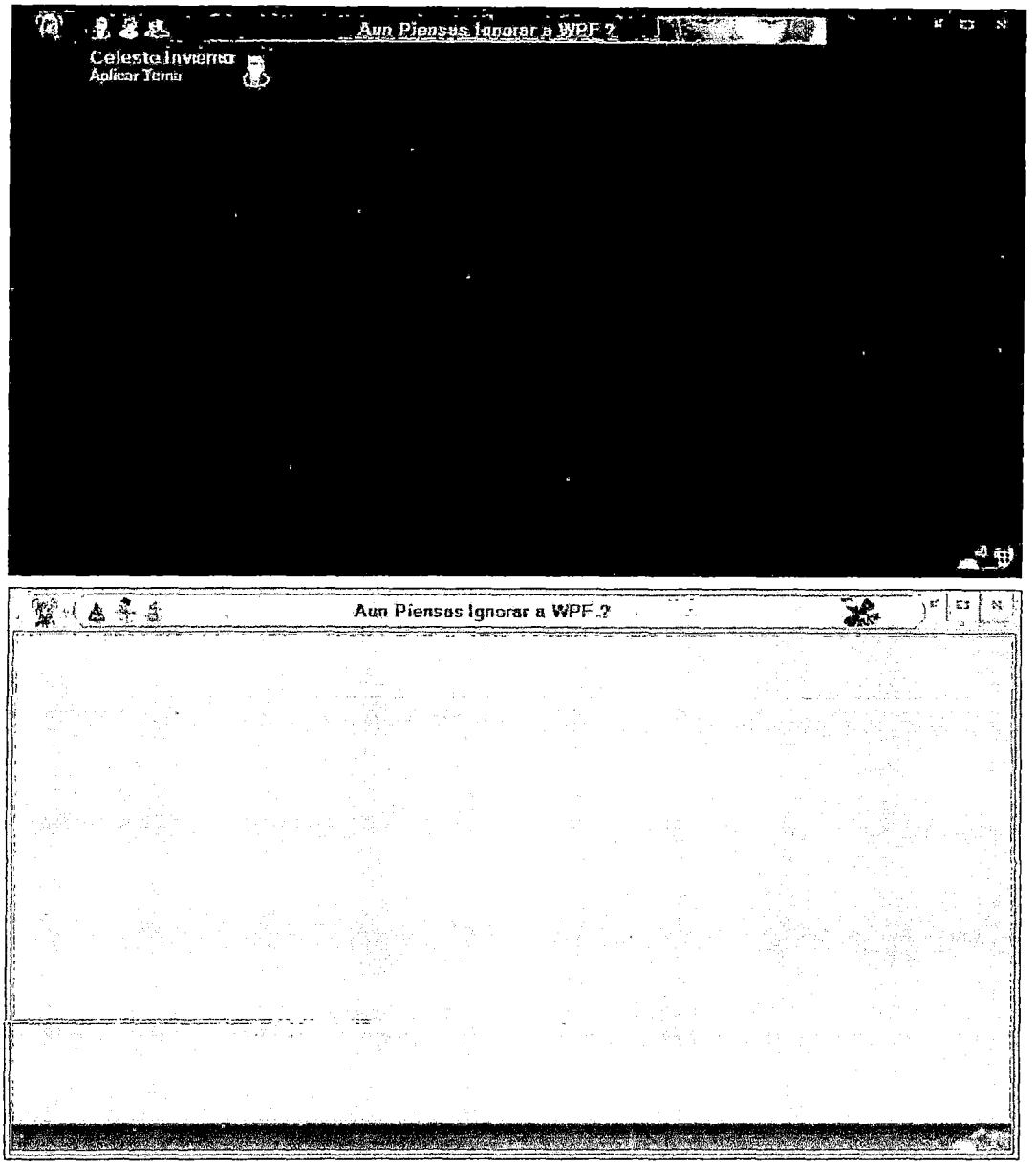


Figura 12: Aspectos de una ventana personalizada en WPF.
Fuente: Fuente propia.

Como se puede apreciar en las imágenes anteriores, se mejoró el aspecto que poseen las ventanas, partiendo (heredando) de un control ya existente que en este caso es Window:

```
namespace WpfMyMetroWindows
{
    public partial class MyFancyWindow : Window
    {
        public MyFancyWindow()...
        ...
    }
}
```

Figura 13: Código del Control de usuario que hereda de un control ya existente que es Window.
Fuente: Fuente Propia.

Los controles de usuario son mucho más fáciles de crear que los controles personalizados, ya que es posible reutilizar los ya existentes. Esto permite crear con facilidad controles con elementos de interfaz de usuario complejos.

Los controles de usuario (User Controls) en .NET heredan de la clase base UserControl y sirven para, usando controles ya existentes extender su funcionalidad o crear grupos de controles reutilizables.

2.2.2.4.1 CARACTERÍSTICAS

- El UserControl, se puede entender como la composición de otros controles para formar uno solo, de hecho en muchos libros, se compara la creación de un User Control con la de una aplicación normal de Windows Forms o WPF, esto es debido a que en términos de programación es lo mismo pues se usa los mismos elementos.
- Los controles de usuario no se pueden ejecutar como archivos independientes. En su lugar, debe agregarlos a las ventanas o formularios, como haría con cualquier otro control.
- En lugar de una directiva Window o Form, el control de usuario contiene una directiva UserControl que define la configuración y otras propiedades.

- Es el control más fácil de crear, debido a que es una composición.
- Su propósito es el de aunar controles en uno solo.

2.2.2.5 CONTROLES PERSONALIZADOS

Los controles personalizados (Custom Controls) en .NET son clases que heredan directamente de Control y que no tienen apariencia ni funcionalidad base definida, sirven para construir un control que no se base en nada existente y definir incluso su apariencia desde cero.

Estos controles, proporcionan una mayor capacidad de maniobra, a la hora de modificar propiedades y de conseguir que el control haga lo que uno desea. Aquí, deberemos crear toda la funcionalidad mediante código.

Aprovechándonos de esta ventaja que Visual Studio .Net nos ofrece, podemos construir uno propio a partir de la clase base: Control, que se adapten a nuestras necesidades de visualización y control, por ejemplo para mostrar datos en movimiento, provenientes de algún proceso industrial, como:

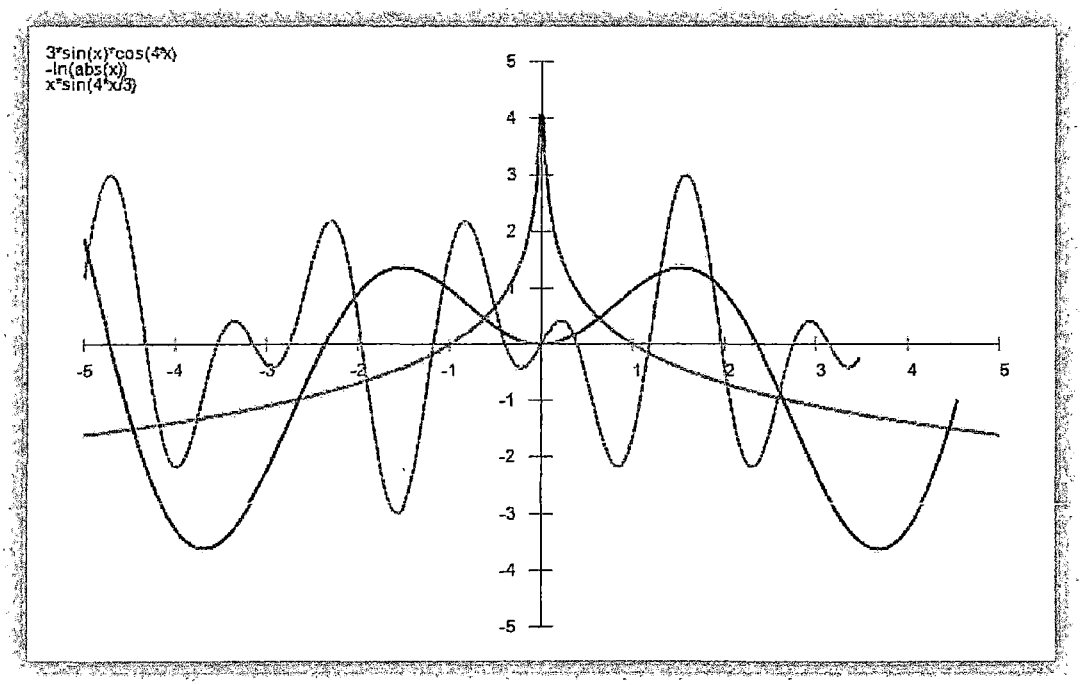


Figura 14: Ejemplo de un control personalizado que muestra datos en movimiento.
Fuente: Fuente propia.

Como se puede apreciar en la imagen anterior, se creó un Control que visualiza datos en movimiento de funciones matemáticas, según estos se van generando; partiendo desde cero pues no se hereda de ningún control existente, sólo de la clase base: Control.

```
namespace WpfPruebas
{
    public class CustContVisualizadorSeñales : Control
    {
        static CustContVisualizadorSeñales()...
        ...

        private void DibujarEjeDeCoordenadas()...
        ...
    }
}
```

Figura 15: Código del componente anterior, que hereda de un Control.
Fuente: Fuente propia.

La creación de un Custom Control, es una tarea mucho más laboriosa, pero que nos permite hacer mucho más, porque ofrece más espacio para jugar con la imaginación, por lo tanto es la mejor opción si se tiene una mejor comprensión del modelo de interfaz en .NET.

2.2.2.5.1 CARACTERÍSTICAS

- Su propósito es mejorar los controles existentes.
- Permite el uso de temas diferentes al de la aplicación.
- Es la mejor opción para las compañías Third-parties, debido a la segunda opción.

2.2.2.6 C-SHARP .NET

C# (pronunciado si sharp en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). En concreto, ha sido diseñado por Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

C# es un lenguaje orientado a objetos sencillo, moderno, amigable, intuitivo y fácilmente legible que ha sido diseñado por Microsoft con el ambicioso objetivo de recoger las mejores características de muchos otros lenguajes, fundamentalmente Visual Basic, Java y C++, y combinarlas en uno sólo, en el que se unan la alta productividad y facilidad de aprendizaje de Visual Basic, la potencia de C++ y la plena orientación a objetos de Java. La orientación a objetos es tal que el propio programa está encapsulado en una clase. A pesar de su corta historia, ha recibido la aprobación del estándar de dos organizaciones: en el 2001 se aprueba el ECMA y en el 2003 el ISO.

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente, diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

Aunque en realidad es posible escribir código para la plataforma .Net en muchos otros lenguajes, como Visual Basic.Net o JScript.Net, C#.Net es el único que ha sido diseñado específicamente para ser utilizado en esta plataforma, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes.

Por esta razón, Microsoft suele referirse a C# como el lenguaje nativo de .Net, y de hecho, gran parte de la librería de clases base de .Net ha sido escrita en este lenguaje.

A continuación se recoge de manera resumida las principales características de C#. No se preocupe si no entiende algunas de ellas, ya que no es indispensable hacerlo para seguir adecuadamente el resto del contenido del taller. Sólo se comentan ahora para que los programadores más experimentados puedan obtener una visión general del lenguaje:

- Dispone de todas las características propias de cualquier lenguaje orientado a objetos: encapsulación, herencia y polimorfismo.

- Adecuación para escribir aplicaciones de cualquier tamaño y propósito: desde las más grandes y sofisticadas como sistemas operativos hasta las más pequeñas funciones.
- Inclusión de principios de ingeniería de software tales como revisión estricta de los tipos de datos, revisión de límites de vectores, detección de intentos de usar variables no inicializadas, y recolección de basura automática.
- Ofrece un modelo de programación orientada a objetos homogéneo, en el que todo el código se escribe dentro de clases y todos los tipos de datos, incluso los básicos, son clases que heredan de System.Object (por lo que los métodos definidos en ésta son comunes a todos los tipos del lenguaje).
- Permite definir estructuras, que son clases un tanto especiales: sus objetos se almacenan en pila, por lo que se trabaja con ellos directamente y no se referencia al montículo, lo que permite accederlos más rápido. Sin embargo, esta mayor eficiencia en sus accesos tiene también sus inconvenientes, fundamentalmente que el tiempo necesario para pasarlas como parámetros a métodos es mayor (hay que copiar su valor completo y no sólo una referencia) y no admiten herencia (aunque sí implementación de interfaces).
- Es un lenguaje fuertemente tipado, lo que significa se controla que todas las conversiones entre tipos se realicen de forma compatible, lo que asegura que nunca se acceda fuera del espacio de memoria ocupado por un objeto. Así se evitan frecuentes errores de programación y se consigue que los programas no puedan poner en peligro la integridad de otras aplicaciones.
- Tiene a su disposición un recolector de basura que libera al programador de la tarea de tener que eliminar las referencias a objetos que dejen de ser útiles, encargándose de ello éste y evitándose así que se agote la memoria porque al programador olvide liberar objetos inútiles o que se produzcan errores porque el programador libere áreas de memoria ya liberadas y reasignadas.
- Incluye soporte nativo para eventos y delegados. Los delegados son similares a los punteros a funciones de otros lenguajes como C++ aunque más cercanos a la orientación a objetos, y los eventos son mecanismos mediante los cuales los objetos pueden notificar de la ocurrencia de sucesos. Los eventos suelen usarse en combinación con los delegados para el diseño de interfaces gráficas de usuario, con lo que se proporciona al programador un mecanismo cómodo para escribir códigos de respuesta a los diferentes eventos que puedan surgir a lo largo de la ejecución de la aplicación. (pulsación de un botón, modificación de un texto, etc.)
- Incluye soporte para la programación funcional que está basado en el cálculo Lambda. La forma más sencilla para conceptualizar las expresiones lambda es pensar en ellas como formas de escribir métodos breves en una línea. Las expresiones lambda se utilizan para crear funciones anónimas las cuales pueden ser asignadas a delegados o a árboles de expresión. Se usan en mayor frecuencia en cálculos matemáticos complejos y en sentencias Linq.

2.2.2.7 XAML[10]

XAML pronunciado zammel acrónimo del inglés Extensible Application Markup Language o Lenguaje Extensible de Marcado para Aplicaciones, es un lenguaje de marcado basado en XML desarrollado por Microsoft, que se usa para definir la presentación visual o interfaz de usuario de las aplicaciones desarrolladas en la plataforma .NET (WPF y Silverlight), del mismo modo que HTML es el lenguaje que se usa para la presentación visual de las páginas Web.

XAML se genera de manera prácticamente automática, al usar las herramientas graficas incluidas en el toolbox de Visual Studio o las herramientas dispuestas para este fin en Expression Blend

Esta optimizado para describir gráficamente interfaces de usuarios visuales "ricas" desde el punto de vista gráfico. En su uso típico, los archivos tipo XAML serían producidos por una herramienta de diseño visual, como Microsoft Visual Studio o Microsoft Expression Blend. El XML resultante es interpretado en forma instantánea por un sub-sistema de despliegue de Windows que reemplaza al GDI+ de las versiones anteriores de Windows.

La intención con XAML es que en un lenguaje declarativo se puedan definir los elementos que compondrán una interfaz de usuario para que estos puedan ser desplegados y conectados con la lógica de la aplicación mediante un motor de presentación que ejecuta sobre .NET Framework conocido como WPF. Algo así como que XAML es el pentagrama y la simbología para expresar la partitura pero WPF es la orquesta para ejecutarla.

XAML propicia separar la definición de las interfaces de usuario de la lógica propia de la aplicación.

XAML es extensible, como su nombre lo indica, lo que significa que los desarrolladores pueden crear sus propios controles y elementos personalizados. Como XAML es por naturaleza una representación textual en XML de los objetos de WPF, puede ser extendido por los desarrolladores por las técnicas de la programación orientada a objetos. De este modo objetos concebidos para la lógica de la aplicación podrán ser expuestos vía XAML para que sean extendidos por los diseñadores visuales.

XAML se basa en una sintaxis bien formada en XML y su fácil extensibilidad, por tanto sigue las reglas sintácticas de XML; a saber: todos los nombres de elementos y de atributos XAML son sensitivos a mayúsculas y minúsculas al menos en el nombre de los Elementos y Atributos. Los valores de los atributos, con independencia de su

[10] Bibliografía – Direcciones Electrónicas[7]

tipo deben escribirse entre comillas dobles. Un único elemento raíz, la utilización de Etiquetas de apertura y cierre, el uso de parámetros, y también el uso de Namespaces, básicamente es XML y la diferencia es que en XAML se tienen espacios de nombres con la definición específica de los elementos de la aplicación.

Los elementos de XAML se interconectan con objetos del entorno común de ejecución para Lenguajes. Los atributos se conectan con propiedades o eventos de esos objetos. Cada elemento XAML tiene por tanto un nombre y puede tener uno o más atributos. Realmente XAML ha sido diseñado para establecer una correspondencia lo más directa posible con el CLR de .NET. Por lo general todo elemento en XAML se corresponde con una clase del CLR de .NET, en particular de WPF y Silverlight, y todo atributo XAML se corresponde con el nombre de una propiedad o de un evento de dicha clase. A la inversa, aunque no todas las clases del CLR están representadas en XAML, aquellas clases que tengan un constructor por defecto y propiedades públicas pueden ser instanciadas declarativamente en código escrito en XAML.

La mayoría de los elementos XAML tienen la característica de que se despliegan visualmente, pueden recibir entrada de teclado y ratón, disparan eventos y saben visualizar el tamaño y posición de sus elementos contenidos (hijos).

La mayoría de los elementos XAML son elementos de interfaz de usuario y derivan de System.Windows.UIElement.

2.2.2.7.1 EXPLICACIÓN BREVE DEL FORMATO XAML

I. ELEMENTOS Y ATRIBUTOS

Para explicar estos conceptos nos basaremos en la siguiente imagen:

```
<Window x:Class="WpfEjemplo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
        <Button  Name="buGenerar" Width="160" Foreground="White" Cursor="Hand">
            <StackPanel Orientation="Horizontal">
                <TextBlock  Text="Generar Script" Foreground="White" FontFamily="Calibri" FontSize="13" />
                <Image Source="Develop.png" Height="16" Width="16" Margin="5,0,0,0" />
            </StackPanel>
        </Button>
    </Grid>
</Window>
```

Figura 16: Código de una Ventana que posee elementos y atributos.
Fuente: Fuente Propia.

- Todo fichero de XAML está formado por un elemento principal que a su vez contiene otros elementos "hijos" y cada uno de esos elementos puede tener atributos. En la figura anterior, el Window que hay al principio es el elemento principal.

- Los Elementos pueden tener atributos por ejemplo, el elemento Window tiene varios atributos como Width, Height, etc., (los que están en color rojo).

- Los Elementos a su vez pueden tener otros elementos en este caso, Grid es el elemento que está "anidado" en el elemento principal. Y si nos fijamos, el elemento Button a su vez tiene otro elemento: StackPanel, que a su vez tiene anidado otros elementos: TextBlock e Image.

- En XAML los Elementos pueden definirse usando el siguiente formato: <Elemento>Contenido</Elemento>, es decir, el contenido de un elemento se indica con el nombre del elemento, después se indican las cosas que contiene y se finaliza usando el formato de fin de elemento, que siempre tiene el formato </Nombre_del_elemento>.

- Los Elementos también pueden tener el formato <Elemento [atributos] />. En este caso, no se usa el cierre del elemento como se vio antes. Y esto es útil si un elemento no tiene un "contenido" específico. Ese contenido puede ser algo que contiene o bien porque contenga otros elementos, pero también puede ser que no tenga un contenido, salvo por lo indicado en las propiedades (o atributos) del elemento. Si ese es el caso, se puede usar en la forma "abreviada". Ese es el caso de los elementos TextBlock e Image del ejemplo anterior. Por supuesto, esos atributos son opcionales, por tanto, si no tiene atributos, es muy posible que el elemento tenga un contenido; aunque no siempre es así, por ejemplo, si se desea indicar un separador en un menú, simplemente se usaría <Separator /> sin más y XAML sabrá que lo que se define es un separador sin atributos ni contenido.

- Si un elemento tiene atributos, estos se indican en la definición del inicio del elemento (si es un elemento con contenido), por ejemplo, el elemento StackPanel tiene varios elementos y un atributo, por tanto ese atributo se define después del nombre del elemento, si hay más atributos en el mismo elemento, estos se separan del anterior en al menos un espacio. Cuando ya se haya definido todos los atributos, se cierra el inicio del elemento con el carácter >.

- Si un elemento no tiene contenido (ni tampoco contiene otros elementos), los atributos se indican antes de los caracteres de cierre, que en el caso de los atributos sin contenido se indica con los caracteres />, el ejemplo se puede apreciar en los elementos TextBlock e Image.

- Los atributos siempre usan el mismo formato, es decir, nombre del atributo, signo igual y entre comillas dobles el valor de ese atributo. Por ejemplo, el atributo Foreground del elemento TextBlock tiene el valor White.

- El hecho de que el código esté coloreado nos permite distinguir mejor cada una de las cosas que definimos, lo de que esté coloreado es solo para que resulte más fácil la lectura del código; en realidad el XML (XAML está basado en XML) no define ninguna norma de coloración pero en el caso de Internet Explorer y Visual Studio, las normas que se siguen son:

  - Los caracteres de apertura y cierre de los elementos se colorean en azul.
  - Los nombres de los elementos se colorean en rojo oscuro (FireBrick).
  - Los atributos se colorean en rojo.
  - Los valores de los atributos se colorean en azul.
  - Los comentarios se colorean en verde.

- XAML al estar basado en XML, sigue algunas reglas de este, por ejemplo no es estricto con los elementos y atributos que usemos, salvo para los casos de nomenclatura, por ejemplo hay ciertos caracteres que no se pueden usar directamente en el código XAML, ni como parte de un nombre de un elemento o de un atributo, por ejemplo, los nombres no deben contener espacios. Y tampoco podemos usar el signo **&**, ya que ese signo se usa para definir otros caracteres, si en un fichero XAML queremos incluir el signo ampersand (**&**) debemos hacerlo en la forma de **&**. Esto es aplicable a cualquier cosa que escribamos para usarlo como XAML, por ejemplo en la documentación XML de nuestro código

II. ELEMENTOS = CONTROLES, ATRIBUTOS = PROPIEDADES

- Se puede afirmar que un elemento XAML es el equivalente de un control.

- En cuanto a los atributos XAML usados con los elementos, se debe pensar en ellos como si fueran propiedades de un control.

Para demostrar que esto que se acaba de decir es cierto, daremos un pequeño ejemplo:

```
<Window x:Class="WpfEjemplo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
      <Button  Name="buGenerar" Content="Generar Script" Width="160" Foreground="White" />
    </Grid>
</Window>
```

Figura 17: Ejemplo de definición de un Botón.
Fuente: Fuente Propia.

El código de la figura anterior, también se podía haber definido de esta otra forma:

```
<Window x:Class="WpfEjemplo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
    <Button  Name="buGenerar" Width="160" Foreground="White">
             Generar Script
    </Button>
</Grid>
</Window>
```

Figura 18: Forma Alterna de definir un Botón.
Fuente: Fuente propia.

En estos dos casos (que son equivalentes), el elemento Button representa un control de tipo System.Windows.Controls.Button (que es el espacio de nombres de los controles WPF) y cada uno de los atributos de ese Elemento/Control son las propiedades del control.

Por ejemplo, el atributo Name se traduce en la propiedad Name del botón. El caso de la Propiedad Content es un caso especial. Esta propiedad es la que contiene el contenido del elemento y como en la mayoría de los casos en XAML, el contenido puede ser cualquier "cosa" (incluso un Elemento/Control), y tal como ocurre en este caso en particular, puede ser hasta una cadena de texto.

Los valores que le podemos asignar a los atributos serán los que a esos atributos (o propiedades) se puedan asignar, y como norma, esos valores se asignan como cadenas de texto, después el compilador se encargará de convertir ese valor de texto en el valor adecuado, y si no lo puede asignar, nos avisará con un error.

2.2.2.8 WINDOWS PRESENTATION FOUNDATION[11]

WPF (Windows Presentation Foundation) es una nueva tecnología para la construcción de la interfaz de usuario en Windows que se ha agregado como pilar fundamental en la liberación del .NET Framework 3.0 y que se ha mejorado y extendido en el .NET Framework 3.5. WPF dentro del desarrollo de aplicaciones utiliza un lenguaje de marcado basado en XML al que se le denomina XAML.

WPF es un sistema de presentación de la próxima generación, para crear aplicaciones cliente de Windows que proporcionen una experiencia impactante para el usuario desde el punto de vista visual. Con WPF puede crear una amplia gama de aplicaciones ricas visualmente, independientes y hospedadas en explorador.

WPF ofrece una amplia infraestructura y potencia gráfica con la que es posible desarrollar aplicaciones visualmente atractivas, con facilidades de interacción que incluyen animación, vídeo, audio, documentos, navegación o gráficos 3D. Separa, con el lenguaje declarativo XAML y los lenguajes de programación de .NET, la interfaz de interacción de la lógica del negocio, propiciando una arquitectura Modelo Vista Controlador para el desarrollo de las aplicaciones.

El núcleo de WPF es un motor de representación basado en vectores e independiente de la resolución que se crea para sacar partido del hardware de gráficos moderno. WPF extiende el núcleo con un conjunto completo de características de desarrollo de aplicaciones que incluye Lenguaje XAML, controles, enlace de datos, diseño, gráficos 2D y 3D, animación, estilos, plantillas, documentos, multimedia, texto, tipografía y mucho más.

2.2.2.8.1 USO DE XAML EN WPF

XAML es un lenguaje de marcado basado en XML que se utiliza para implementar la apariencia de una aplicación mediante declaración. Se suele utilizar para crear ventanas, cuadros de diálogo, páginas y controles de usuario, así como para rellenarlos con controles, formas y gráficos. En tiempo de ejecución, WPF convierte los elementos y atributos definidos en el marcado en instancias de clases de WPF.

WPF dentro del desarrollo de aplicaciones, utiliza un lenguaje de marcado basado en XML al que se le denomina XAML, este lenguaje de marcado es utilizado también por Silverlight, WF (Workflow Foundation) y por WCF (Windows Communication Foundation), en nuestro caso nos ocupa WPF, por lo que encontraremos que la

[11] Bibliografía – Textos [6]

definición de la ventana, y los controles de una aplicación WPF para Windows estará definida completamente en XAML con etiquetas propias para esta tarea.

Hablando de aplicaciones WPF para Windows, el elemento raíz de la ventana es Window; además se incluye la declaración de ciertos espacios de nombres necesarios para la presentación de la interfaz de usuario, aquí se muestra un ejemplo

```
<Window x:Class="WpfEjemplo.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" >
     <Grid>

    </Grid>
</Window>
```

Figura 19: Ejemplo de definición de una ventana
Fuente: Fuente propia

Podemos ver que se utiliza la etiqueta Window y como parámetros se incluyen la declaración de dos espacios de nombres y un parámetro que indica la clase con la que se asocia la funcionalidad de la ventana. Ahora bien, las propiedades de la ventana se definen de igual manera dentro de esta etiqueta, el alto y ancho de la venta, así como su título, eventos y otras propiedades serán puestos como parámetros en esta etiqueta, como se muestra a continuación:

```
<Window x:Class="WpfEjemplo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Mi Primer Ejemplo"
        Height="350" Width="525"
        Background="SteelBlue"
        WindowStartupLocation="CenterScreen">

</Window>
```

Figura 20: Ejemplo de definición de una ventana que muestra algunas propiedades.
Fuente: Fuente propia.

En este bloque se puede observar que se agregó el título y las propiedades Height y Width, cabe hacer notar que los valores de las propiedades están escritos entre comillas y siempre serán tomados como texto. Este código se guarda en un archivo con extensión .xaml y está ligado a un archivo .cs o .vb según sea C# o Visual Basic,

en el cual se escribirá la funcionalidad de la ventana utilizando el lenguaje que más conocemos.

Cuando se crea una ventana WPF en Visual Studio se agrega esta declaración y además se agrega dentro del elemento Window un control denominado Grid, que no debemos confundir con el DataGrid que se utiliza para otro fin. Este control Grid será el que se encarga de mostrar los controles en el orden que nosotros definamos y puede ser utilizado análogamente como una tabla de HTML, en la cual se pueden definir columnas y renglones y confinar los controles a una celda específica. Veamos primeramente cómo se muestra la ventana al ser creada por primera vez:

```
<Window x:Class="WpfEjemplo.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Mi Primer Ejemplo"
        Height="350" Width="525"
        Background="SteelBlue"
        WindowStartupLocation="CenterScreen">
    <Grid>

    </Grid>
</Window>
```

Figura 21: Ejemplo de definición de una ventana, con un contenedor de tipo Grid.
Fuente: Fuente propia.

En este ejemplo el elemento Grid, no tiene definida ninguna propiedad por lo que el ancho y alto será limitado por la dimensión definida en la ventana, esto es 350 x 525, y no se han definido propiedades por consiguiente el Grid se verá como el área de trabajo para el diseño de la venta, por esta razón no define ninguna propiedad. Otra de las razones importantes de utilizar un Grid es porque de no hacerlo solo podríamos declarar un control dentro de la ventana, ya que entre la etiqueta de apertura y cierre, que corresponde a la propiedad Content (contenido) de la ventana, solo puede contener un control, en cambio, el Grid, puede contener a varios controles.

La siguiente imagen muestra un texto "Hola Mundo" dentro de un contenedor de tipo Grid.

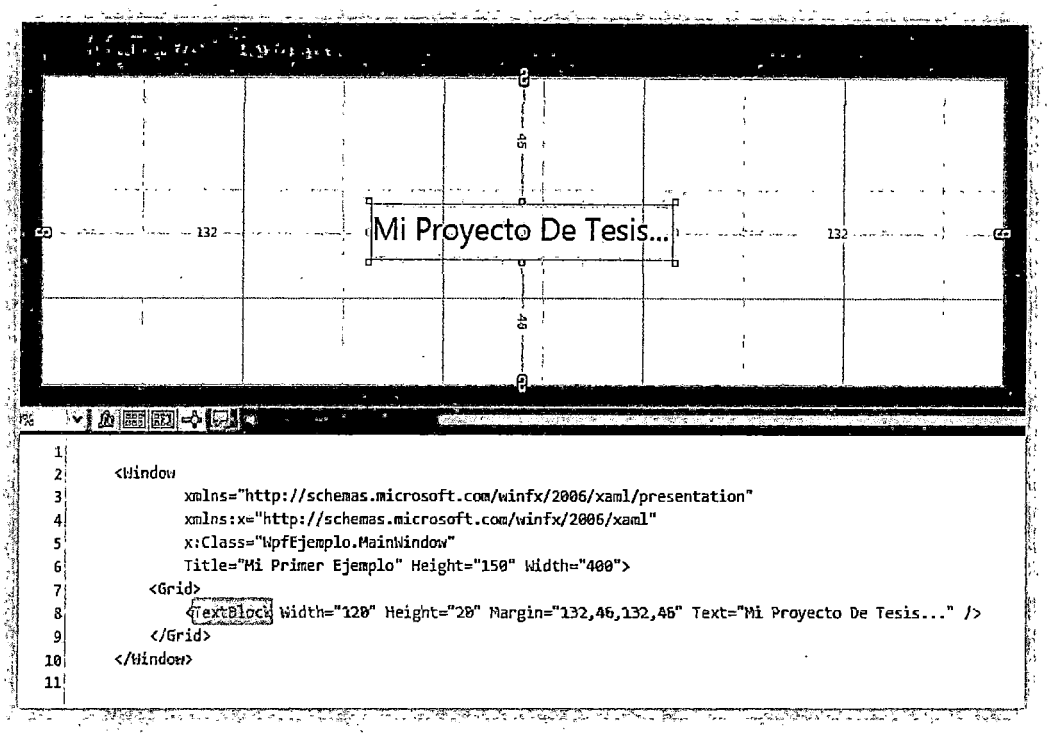


Figura 22: Ejemplo de inserción de elementos dentro de un contenedor de tipo Grid.
Fuente: Fuente propia.

Como se puede observar, el TextBlock tiene definidas algunas propiedades; Width, Height, Margin y Text. La propiedad Text no tiene mucho que decir, pues es la que se encarga de mostrar el texto. Las otras propiedades tienen sus particularidades.

En general las propiedades Width y Height están presentes en la mayoría de los controles, y sirve para indicar el ancho y la altura del mismo, si no se definen, o en algunos casos se define el valor Auto, el control tomará la anchura y altura del contenedor, en este caso del Grid.

La propiedad Margin, define el margen del control, esto es, la distancia que guarda un control con los límites de su contenedor. La propiedad Margin, se escribe con cuatro valores separados por comas que definen la distancia entre los límites de su contenedor en el siguiente orden: margen izquierdo, margen superior, margen derecho, margen inferior. En este caso, el TextBlock se ajustará al Grid manteniendo la distancia de 132 pixeles entre el extremo derecho e izquierdo, además, mantendrá una distancia de 46 pixeles con la parte superior, y podríamos suponer que tendría la misma distancia al inferior.

2.2.2.8.2 INTERACCION ENTRE PROGRAMADOR Y DISEÑADOR MEDIANTE EL USO DE XAML

Lo que hace más valioso al enfoque de WPF basado en XAML es que al quedar la interfaz de usuario especificada con un lenguaje de marcas y declarativo, como es XAML, la hace legible y más fácilmente "retocable" por los humanos y lo que es más importante, fácilmente susceptible de ser analizada y procesada por herramientas. Esto ofrece a propios y terceros la posibilidad de crear herramientas de diseño visual que soporten XAML.

La separación entre código XAML para expresar la apariencia y código de un lenguaje .NET como C# para expresar la lógica de la aplicación, facilitará que los diseñadores con capacidades artísticas (posiblemente con experiencias en los lenguajes de marcas como HTML pero poco habilidosos para la programación), puedan integrar mejor y con más efectividad su trabajo con los programadores, concentrados en la lógica de la aplicación (que por lo general son muy rústicos a la hora de diseñar una atractiva apariencia). Con ello se facilitará poder modificar la interfaz visual sin tener que tocar la lógica de la aplicación (incluso sin ver código C#) y viceversa.

Una de las cosas que más resalta WPF, es la separación entre la apariencia o interfaz de usuario de lógica de la aplicación y lo lleva al extremo de ofrecer una herramienta para diseñadores gráficos que se integra en el proceso de desarrollo, por lo que el desarrollador simplemente tiene que ir añadiendo su código sin tener que tocar nada de la interfaz y el diseñador lo mismo pero en su terreno.

Para que desarrollador y diseñador puedan trabajar en un mismo proyecto con estas herramientas tiene que existir un nexo de unión, este nexo es XAML, un lenguaje al estilo de XML y que es entendido por Visual Studio del lado del desarrollador y Expression Blend por el lado del diseñador.

Cuando el programador crea una interfaz gráfica se concentra en los elementos desde el punto de vista lógico y en cómo se comunican entre sí y con los datos de la aplicación, los ficheros generados son directamente accesibles con Microsoft Expression Blend, allí el diseñador encuentra una aplicación con la que es sencillo cambiar el aspecto visual de los elementos, aplicar efectos y diseñar animaciones.

Anteriormente la forma de comunicación entre programador y diseñador era un bosquejo que creaba el diseñador de cómo debía más o menos aparecer la interfaz del programa y el desarrollador intentaba llevarla a cabo con algún lenguaje de programación, como se aprecia en la figura siguiente:

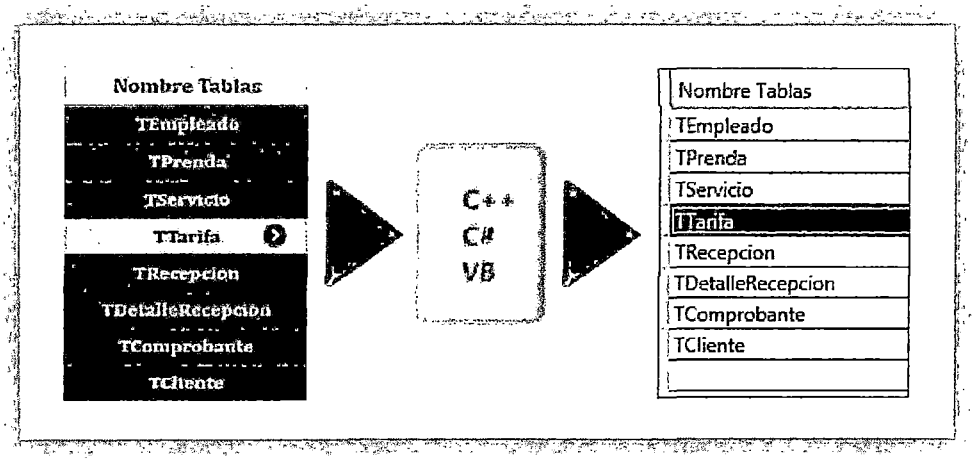


Figura 23: Intento de conseguir el trabajo del Diseñador por parte de un Programador.
Fuente: Fuente Propia.

Ahora el diseñador cuando crea una interfaz, es ésa la que aparecerá finalmente sin adaptaciones de ningún tipo, como se puede apreciar en la figura siguiente:

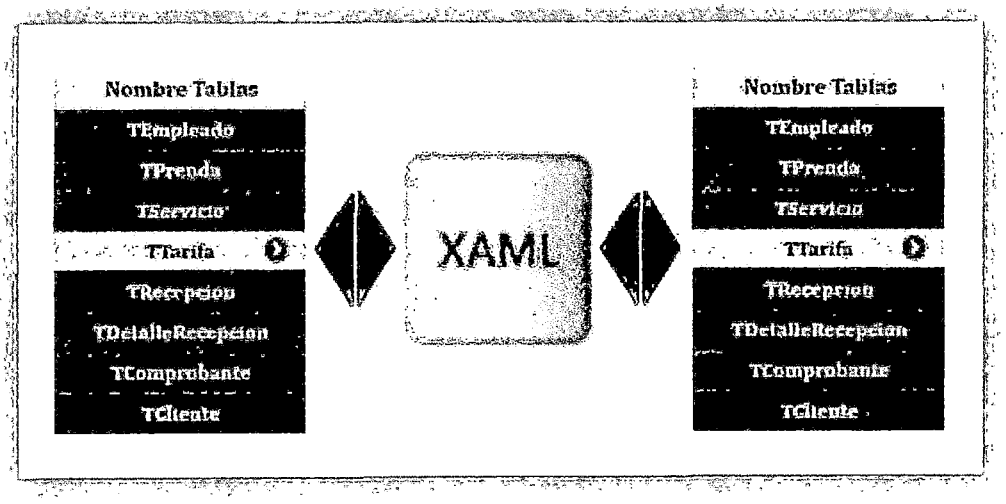


Figura 24: Mismo resultado obtenido por parte del Programador, al usar Expression Blend.
Fuente: Fuente propia.

2.2.2.8.3 CARACTERÍSTICAS DE WPF

A continuación se describe las características más resaltantes de esta herramienta de desarrollo:

DIRECT 3D

WPF pone énfasis en los gráficos vectoriales porque esto permite a la mayoría de los controles y elementos ser escalados sin pérdida de calidad o pixelizacion, aumentando así la accesibilidad.

Todos los gráficos, incluyendo los elementos de escritorio como Windows, se representan usando Direct3D. Esto proporciona una vía para mostrar gráficos más complejos y temas personalizados, a costa de GDI+ una gama más amplia de apoyo y tematización uniforme de control. También permite descargar algunas de las tareas de gráficos a la GPU (*graphics processing unit*). Esto puede reducir la carga de trabajo de la CPU.

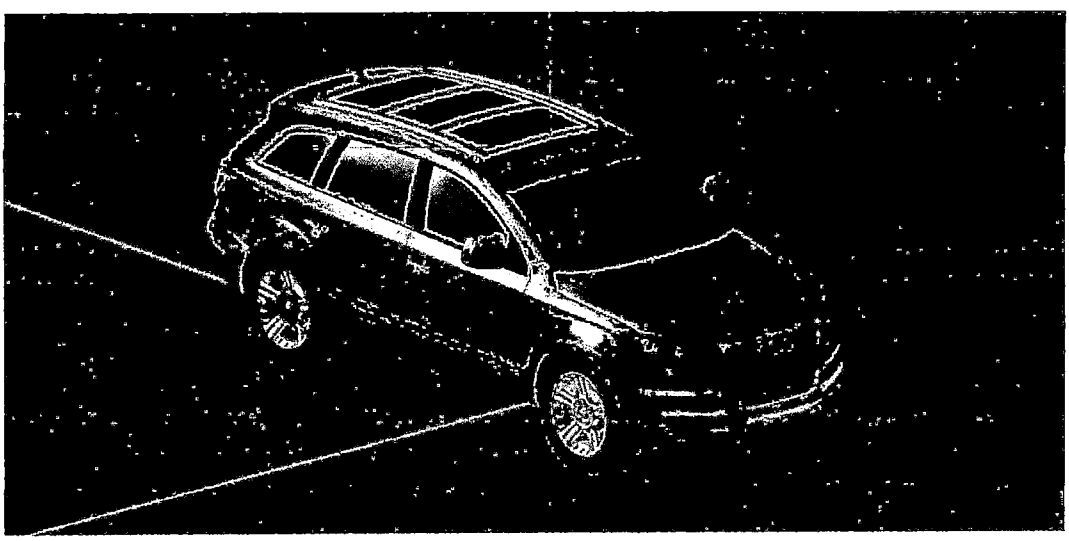


Figura 25: Inclusión de elementos 3D en un proyecto WPF usando un contenedor ViewPort3D
Fuente: http://stackoverflow.com/questions/3127753/displaying-3d-models-in-wpf
Acceso: 25 de Marzo del 2013

DATA BINDING

El enlace a datos permite que una interfaz de usuario refleje visualmente lo qué está pasando por detrás del telón con los datos de la aplicación, y a la inversa también ya que la interacción que se realiza con la interfaz queda reflejada en los datos de la aplicación.

La mayoría de las aplicaciones se crean para proporcionar recursos a los usuarios que les permitan ver y editar los datos. Después de obtener acceso a los datos y de cargarlos en los objetos administrados de una aplicación, comienza la tarea ardua de las aplicaciones. En esencia, esto implica dos cosas:

- Copiar los datos desde los objetos administrados en los controles, donde los datos se pueden mostrar y editar.
- Asegurarse de que los cambios realizados en los datos mediante los controles se vuelvan a copiar en los objetos administrados.

Para simplificar el desarrollo de aplicaciones, WPF proporciona un motor de enlace de datos que realiza estos pasos automáticamente (evita tener que programar la plomería necesaria para mantener sincronizados los datos en ambos extremos del enlace). La unidad que constituye el núcleo del motor de enlace de datos es la clase Binding, encargada de enlazar un control (el destino de enlace) a un objeto de datos (el origen de enlace).

La clase Binding, representa a los objetos que permiten enlazar una fuente de datos con un destino de datos. Ésta relación se muestra en la ilustración siguiente:

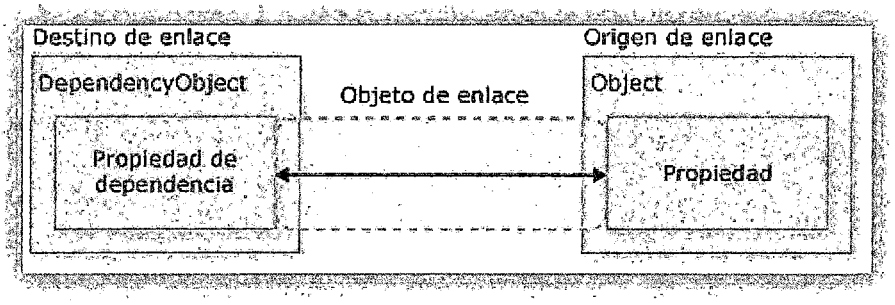


Figura 26: Funcionamiento de las propiedades de dependencia
Fuente: http://msdn.microsoft.com/en-us/library/ms752347.aspx
Acceso: 25 de Marzo del 2013

A un enlace se le puede indicar el momento y la dirección en que debe mantener el enlace. Esto se logra mediante la propiedad Mode de la clase Binding que es de tipo BindingMode.

BindingMode posee varios modos como son "una vez" (OneTime), "unidireccional" (OneWay), "unidireccional invertido" (OneWayToSource) y "bidireccional" (TwoWay). Estos, como sus nombres indican, permiten activar el enlace solo la primera vez, solo cuando se actualiza la fuente, solo cuando se actualiza el destino, o cuando se actualiza cualquiera de los dos, respectivamente. Finalmente el valor Default permite decidir, al sistema de propiedades de WPF, el comportamiento del enlace. Este es el valor que toma la propiedad Mode del enlace cuando no se especifica un valor y que se traduce en la práctica en OneWay o TwoWay en dependencia de la propiedad destino del enlace. La siguiente Figura ilustra el comportamiento de los diferentes valores de la propiedad Mode en el uso de enlaces:

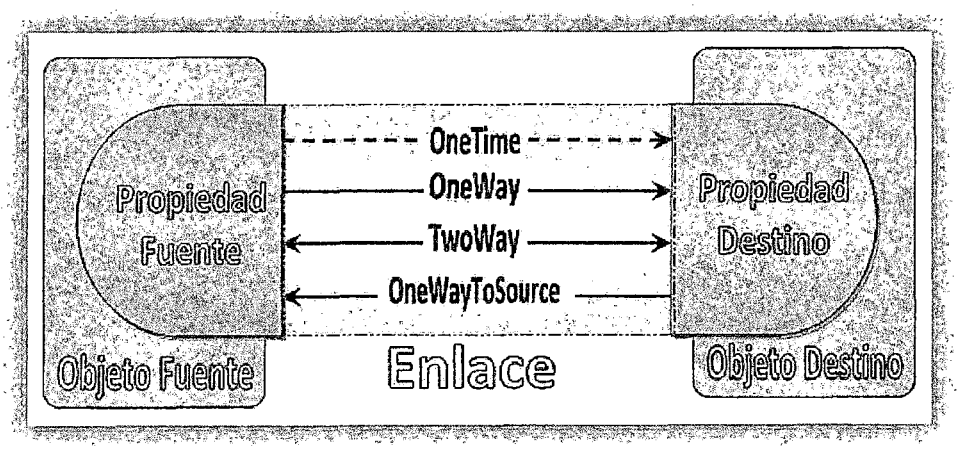


Figura 27: Modos de enlazar elementos.
Fuente: http://msdn.microsoft.com/en-us/library/ms752347.aspx
Acceso: 25 de Marzo del 2013

En la siguiente figura se ilustra el funcionamiento básico de la clase Binding. El objeto Binding está representado en la figura como la línea que une la fuente y el destino, en una o ambas direcciones. A su vez, la fuente y el destino pueden ser de diferentes proveedores de datos como Gestores de datos, objetos de negocio, XML, Colecciones, Controles visuales que permiten la interacción con la aplicación y prácticamente cualquier cosa que contenga datos.

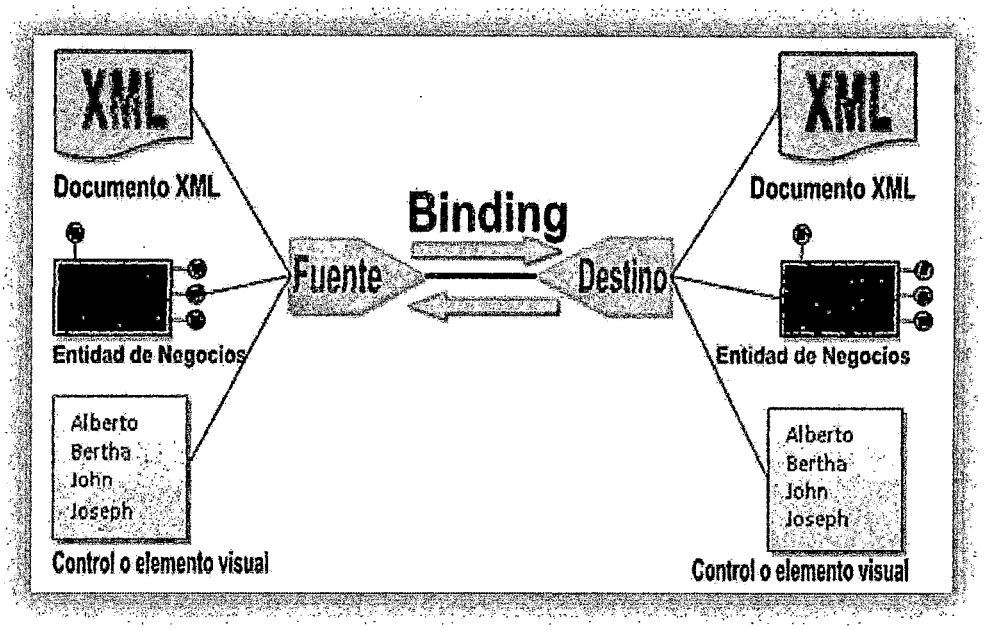


Figura 28: Esquema de uso de objetos de la clase Binding
Fuente: Bienvenido Al Curso De WPF p. 165-166
Acceso: 25 de Marzo del 2013

MEDIA SERVICES

WPF proporciona un sistema integrado para la creación de interfaces de usuario con elementos multimedia comunes, como imágenes vectoriales y de mapa de bits, tipografía, efectos, audio y vídeo, etc. WPF también proporciona un sistema de animación y un sistema de renderizado en 2D/3D. A continuación trataremos de resaltar las más importantes:

I. ***Texto y Tipografía;*** Para facilitar una representación de texto de gran calidad, WPF ofrece las características siguientes:

- Compatibilidad con fuentes OpenType.
- Alto rendimiento que saca partido de la aceleración de hardware.
- Integración de texto con multimedia, gráficos y animación.
- Compatibilidad con fuentes internacionales y mecanismos de reserva.

A modo de demostración de la integración del texto con gráficos, en la ilustración siguiente se muestra un ejemplo que aplica decoraciones a un texto.

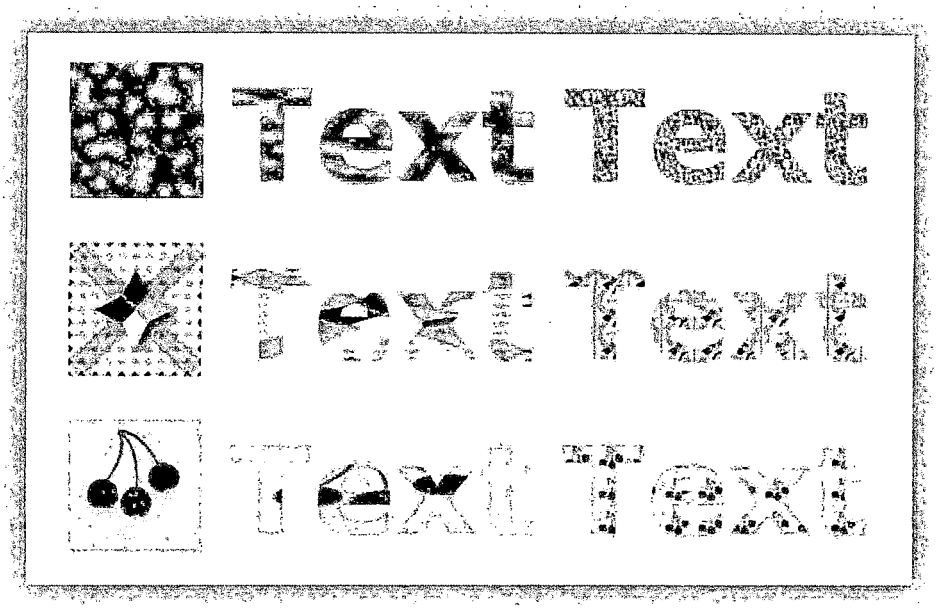


Figura 29: Ejemplo de estilos aplicados a los textos.
Fuente: http://msdn.microsoft.com/en-us/library/ms752347.aspx
Acceso: 25 de Marzo del 2013

II. ***Formas 2D;*** WPF proporciona un conjunto extenso de primitivas de forma para gráficos en 2D, dibujados mediante vectores, junto con un conjunto integrado de pinceles, lápices, geometrías, y transformaciones, como se puede apreciar en la imagen siguiente:

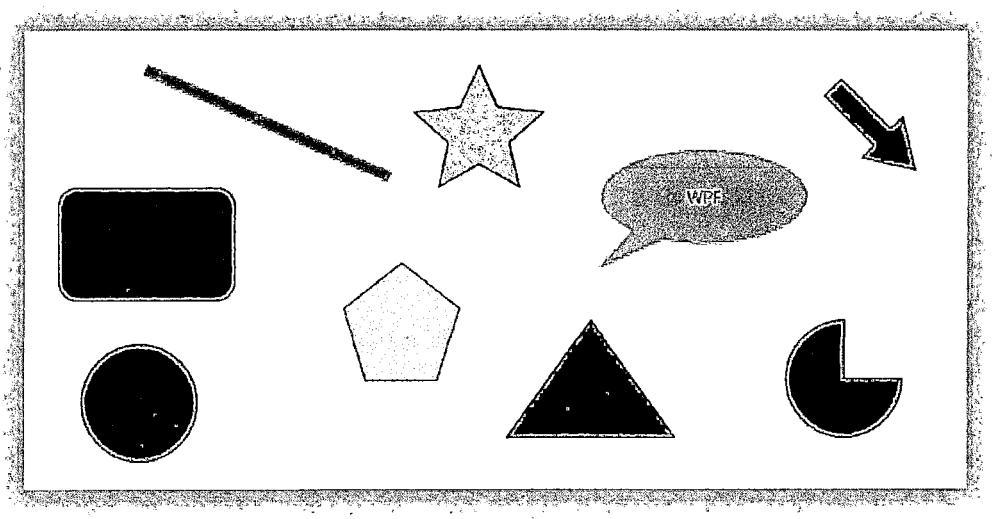


Figura 30: Ejemplo de figuras básicas en WPF
Fuente: Fuente propia

III. ***Efectos 2D;*** Un subconjunto de las funciones 2D de WPF son los efectos visuales, tales como degradados, mapas de bits, dibujos, pintar con vídeos, rotación, ajuste de escala y sesgo, etc. Todas ellas se aplican mediante pinceles; en la siguiente ilustración se muestran algunos ejemplos:

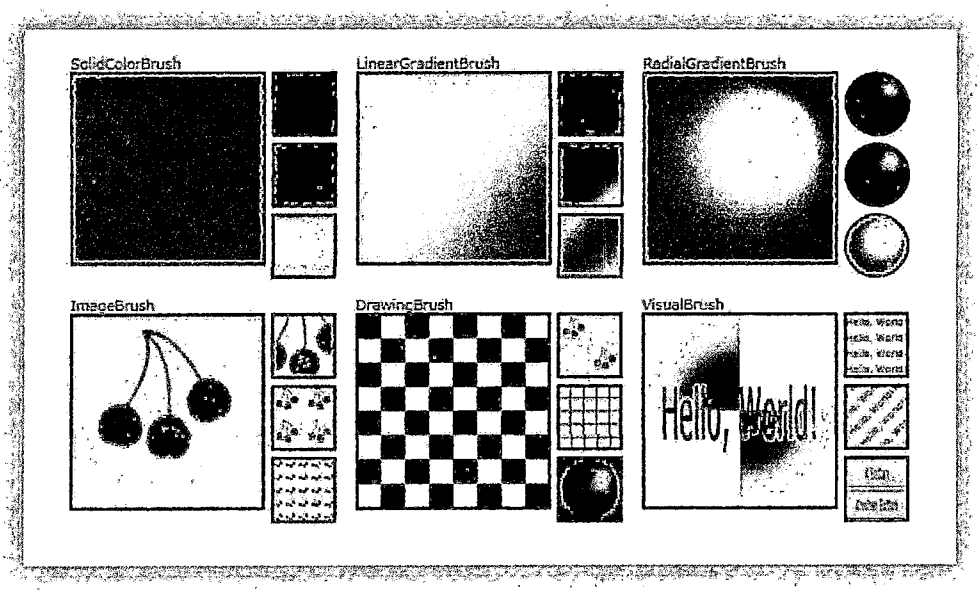


Figura 31: Ejemplos de aplicación de efectos visuales en WPF
Fuente: http://msdn.microsoft.com/en-us/library/ms752347.aspx
Acceso: 25 de Marzo del 2013

IV. ***Documentos;*** WPF incluye compatibilidad nativa para trabajar con tres tipos de documentos: documentos dinámicos, documentos estáticos y documentos XML Paper Specification (XPS). WPF también proporciona servicios para crear ver administrar agregar empaquetar e imprimir documentos.

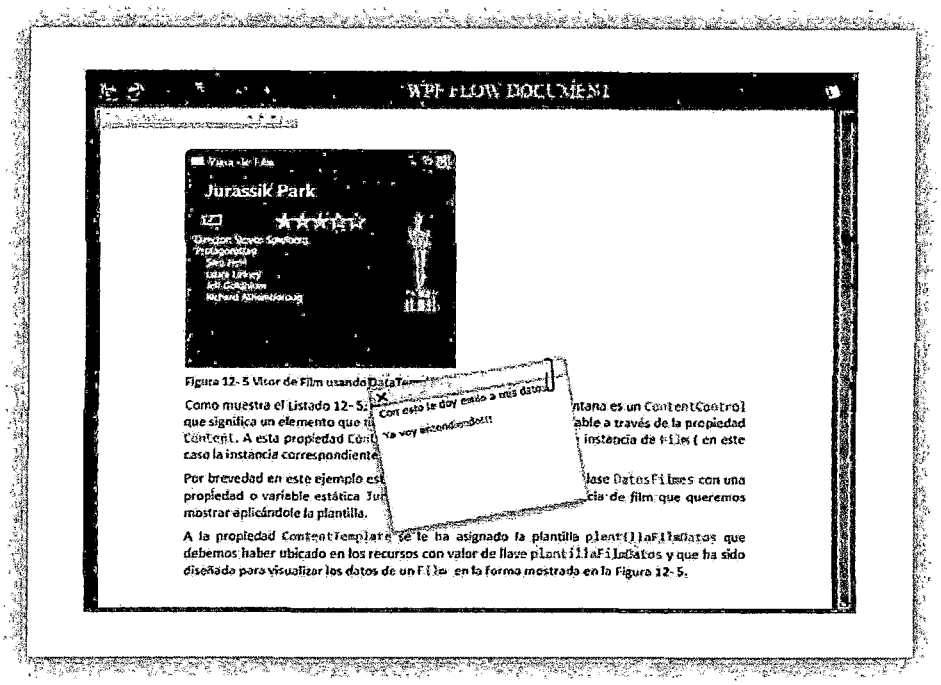


Figura 32: ***Inserción de documentos en*** **WPF.**
Fuente: Bienvenido Al Curso De WPF p. 165-166.
Acceso: 25 de Marzo del 2013

V. ***Representación 3D;*** La funcionalidad de 3D en Windows Presentation Foundation permite a los desarrolladores dibujar, transformar y animar gráficos 3D en el marcado y en el código de procedimiento. Los desarrolladores pueden combinar gráficos 2D y 3D para crear controles enriquecidos, proporcionar ilustraciones complejas de datos o mejorar la experiencia del usuario de la interfaz de una aplicación.

Las capacidades 3D de WPF son un subconjunto del conjunto completo de características proporcionado por Direct3D. La compatibilidad con 3D en WPF no se ha diseñado para proporcionar una plataforma completa de desarrollo de juegos.

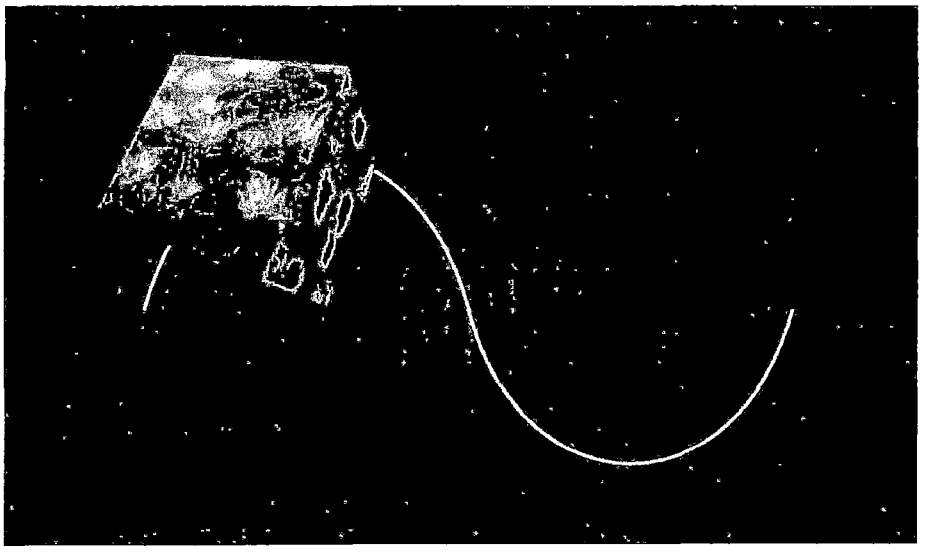


Figura 33: Ejemplo de representación y animación en 3D
Fuente: Paquete de ejemplos de Microsoft, incluidas en Visual Studio.

VI. ***Audio y Video;*** WPF admite por defecto los formatos de vídeo WMV, MPEG y algunos archivos AVI, pero como por debajo ejecuta Windows Media Player, WPF puede usar todos los codecs instalados para el mismo.

El control MediaElement presente en la barra de herramientas de WPF, es capaz de reproducir vídeo y audio y presenta la flexibilidad suficiente para constituir la base de un reproductor multimedia personalizado, como la que se muestra a continuación:

Figura 34: Aplicación de un reproductor de música realizado en WPF.
Fuente: http://browse.deviantart.com/art/Unnamed-Media-Center-v0-1-298423001

Figura 35: Aplicación de un reproductor de videos, realizado en WPF.
Fuente: http://browse.deviantart.com/art/Unnamed-Media-Center-v0-1-298423001

ESTILOS Y PLANTILLAS

Los estilos nos permiten personalizar la apariencia de la interfaz de usuario de modo similar y más amplio que lo que por ejemplo se puede hacer con los estilos en Microsoft Word. Los estilos permiten crear configuraciones que indiquen características visuales de controles, figuras, texto, etc. para que luego puedan ser aplicadas a diferentes elementos.

Las plantillas (*templates*) son un recurso que nos facilitará expresar patrones de código a aplicar a datos y a controles propiciando la reutilización y aumentando la productividad y flexibilidad

- En WPF puede definir el aspecto de un elemento directamente, a través de sus propiedades o indirectamente a través de una plantilla o estilo. En su forma más simple un estilo es una combinación de valores de propiedades que se pueden aplicar a un elemento de interfaz de usuario con un atributo de la propiedad individual. Las plantillas son un mecanismo alternativo para la definición de interfaz de usuario para partes de la aplicación de WPF.
- Todos los controles de WPF tienen por defecto una plantilla que define su árbol visual. La plantilla predeterminada es creado por el autor del control y puede ser sustituible por otros desarrolladores y diseñadores.

ANIMACIONES

Las animaciones pueden hacer que una interfaz de usuario sea más vistosa y práctica. Con anterioridad a WPF, los desarrolladores de Microsoft Windows tenían que crear y administrar sus propios sistemas de temporización o utilizar bibliotecas personalizadas especiales. WPF incluye un sistema de control de tiempo eficaz que se expone a través del código administrado y del Lenguaje XAML. La animación WPF facilita la animación de controles y otros objetos gráficos.

WPF controla todo el trabajo de administración del sistema de temporización y de actualización de la pantalla que se produce en segundo plano. Proporciona clases de temporización que permiten centrarse en los efectos que se desea crear, en lugar de la mecánica para lograr esos efectos. WPF también facilita la creación de sus propias animaciones exponiendo clases base de animación de las que se pueden heredar sus clases, para generar animaciones personalizadas.

En WPF las animaciones pueden usarse casi en cualquier parte, lo que incluye estilos y plantillas de control.

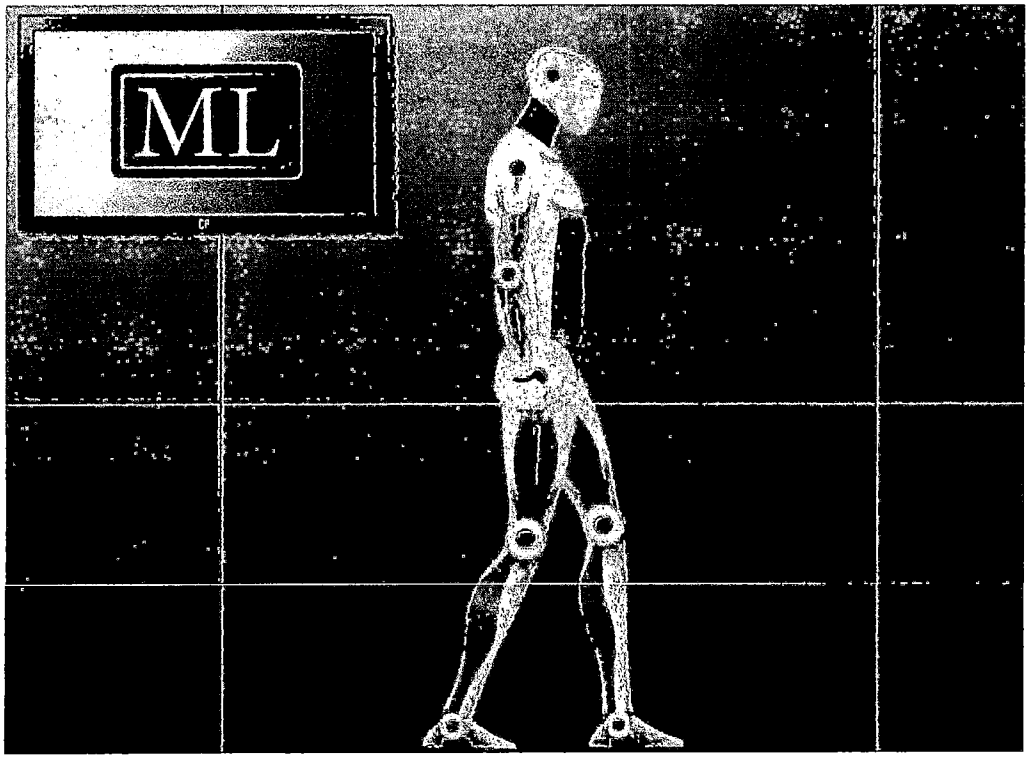


Figura 36 Animación del movimiento de un humano en WPF
Fuente http://www.codeproject.com/Articles/142249/XAML-Man
Acceso: 25 de Marzo del 2013

PROPIEDADES DE DEPENDENCIA[12]

Cuando se empieza a desarrollar una aplicación en WPF, pronto nos toparemos con Propiedades de Dependencia (DependencyProperties). Se ven muy similares a las propiedades normales de .NET, pero el concepto detrás es mucho más complejo y de gran alcance.

WPF introduce un nuevo uso de las propiedades mucho más inteligente y sofisticado, estas son las ***DependencyProperties,*** *que* pueden notificar los cambios que en élla se producen.

Una "DependencyProperty" es un nuevo tipo de propiedad que introduce WPF para facilitar características como Styling, DataBinding, Animación etc. El motivo principal por el cual WPF añade esta "inteligencia" a las propiedades, es para habilitar funcionalidad enriquecida en la declaración del marcado XAML. Un botón por ejemplo, tiene más de 90 propiedades públicas, las propiedades pueden ser configuradas directamente en XAML sin necesidad de código procedural, pero sin la existencia de "DependencyProperties" se tendría que escribir código .NET para realizar tareas, como por ejemplo modificar el color de fondo de un botón cuando el mouse pasa por encima del mismo.

Mediante las "DependencyProperties" podemos habilitar este tipo de comportamientos directamente en código XAML sin necesidad de escribir una sola línea de código .NET.

La principal diferencia es que el valor de una propiedad normal de .NET se lee directamente de un miembro privado de su clase, mientras que el valor de una Propiedad de Dependencia se resuelve de forma dinámica cuando se llama al método GetValue() que hereda de DependencyObject.

Las propiedades de dependencia ofrecen la instalación de cañerías para la resolución de valor de la propiedad, notificación de cambio, el enlace de datos, diseño, validación, etc. de las propiedades expuestas en Windows Presentation Foundation.

[12] Bibliografía – Direcciones Electrónicas [5],[6]

```
public static readonly DependencyProperty ActualValorProperty =
DependencyProperty.Register("ActualValor", typeof(double), typeof(controltermometro),
new FrameworkPropertyMetadata(388.0, new PropertyChangedCallback(OnPropertyChange)));

public double ActualValor
{
    get
    {
        return (double)GetValue(ActualValorProperty);
    }
    set
    {
        SetValue(ActualValorProperty, value);
    }
}
```

Figura 37: Ejemplo de una propiedad de dependencia
Fuente: Fuente propia

2.2.2.9 MICROSOFT EXPRESSION BLEND

Microsoft Expression Blend es una completa herramienta de diseño profesional, para crear interfaces de usuario atractivas y sofisticadas, para aplicaciones de Microsoft Windows que se basan en Windows Presentation Foundation (WPF), y para las aplicaciones Web que se basan en Microsoft Silverlight. Permite a los diseñadores centrarse en la creatividad y a los desarrolladores en la programación.

Permite crear interfaces gráficas para Web y para aplicaciones de Escritorio que mezclan características de estos dos tipos de aplicaciones. Puede insertar imágenes, clips de audio y vídeo y personalizar controles de bibliotecas de SDK o terceros. Lo que se verá en la superficie de diseño de Expression Blend es exactamente lo que verán los usuarios cuando ejecuten la aplicación.

Visual Studio siempre ha estado dirigido a los desarrolladores y muchos diseñadores no se sentían cómodos usándolo, pues debían lidiar con multitud de elementos que les eran totalmente ajenos. Para remediar esto, en Junio de 2006 Microsoft lanzo una nueva apuesta al mercado: Expression Blend. Este nuevo software permite a diseñadores y desarrolladores trabajar juntos en un proyecto Silverlight o WPF, pero cada uno usando una herramienta específica para sus necesidades. De esta forma, Expression Blend elimina todo el contenido superfluo para un diseñador y le permite centrarse en su trabajo: diseñar.

2.2.2.9.1 CARACTERÍSTICAS

A continuación se citaran las características más resaltantes de esta herramienta:

- Posee un conjunto completo de herramientas de dibujo vectorial, que incluye herramientas tridimensionales (3D) y de texto. Compatibilidad con elementos en 3D y multimedia para mejorar las experiencias de los usuarios.
- Interoperabilidad con Visual Studio para ayudar a los diseñadores y programadores a colaborar de forma más estrecha y eficaz como un equipo.
- Una interfaz visual moderna y fácil de usar con paneles acoplables y menús contextuales en objetos.
- Animación en tiempo real.
- Capacidades de importación de material gráfico de Expression Design.
- Capacidades de importación de sitios de Expression Encoder.
- Opciones de máscara y personalizaciones avanzadas, flexibles y reutilizables para diversos controles comunes.
- Eficaces puntos de integración de orígenes de datos y recursos externos.
- Vistas de marcado y diseño en tiempo real y mucho más.
- SketchFlow, un nuevo conjunto de características para crear prototipos que son aplicaciones reales de WPF o Silverlight.

2.2.2.9.2 MODO DE TRABAJO DE MICROSOFT EXPRESSION BLEND

En Expression Blend, se puede diseñar la aplicación visualmente, se pueden dibujar formas, trazados y controles en la mesa de trabajo y a continuación se puede modificar su apariencia y comportamiento. Puede importar imágenes, vídeo y sonido, en las aplicaciones basadas en Windows, también puede importar y cambiar objetos 3D. Se puede crear guiones gráficos que animen los elementos visuales del diseño y opcionalmente activar esos guiones gráficos cuando los usuarios interactúan con la aplicación. Cuando trabaja en aplicaciones basadas en Windows o en Silverlight, puede rediseñar las plantillas que se aplican a controles básicos para que la aplicación tenga un aspecto y un comportamiento exclusivos.

Se puede importar gráficos y recursos del lenguaje de marcado de aplicaciones extensible (XAML), generados por Microsoft Expression Design en el proyecto de Expression Blend. Además, puede importar proyectos multimedia de Silverlight creados en Microsoft Expression Encoder para agregar nuevas características o elementos visuales al proyecto o para modificar la plantilla del reproductor multimedia que se puede reutilizar en Expression Encoder.

En Microsoft Expression Web, puede importar sitios web de Silverlight y archivos de una aplicación Silverlight compilados en un proyecto nuevo o existente y a continuación publicar el trabajo.

2.2.2.9.3 INTERACCION

Si se usa Expression Blend, los diseñadores y programadores pueden trabajar en el mismo proyecto simultáneamente y sin molestarse. Uno de los puntos fuertes de Blend es la interacción que existe entre éste y Visual Studio ya que se puede crear un nuevo proyecto en Visual Studio y después con un simple click de ratón editarlo en Expression Blend, o al revés, crearlo en Expression Blend y editar su código en Visual Studio. Todo esto al mismo tiempo de manera sincronizada, sencilla y sin complicaciones.

Microsoft Visual Studio funciona perfectamente con Expression Blend para mantener la sincronización al modificar archivos de manera simultánea en Visual Studio y Expression Blend.

2.2.2.9.4 DIBUJAR EN MICROSOFT EXPRESSION BLEND

Cuando se realiza una aplicación en la cual se desea insertar formas complejas y algunos controles personalizados, resulta bastante complicado realizar todo el trabajo posible directamente escribiendo XAML en Visual Studio, en estos casos particularmente es preferible usar Expression Blend, que resulta ser mucho más productivo, como se muestra en la imagen siguiente.

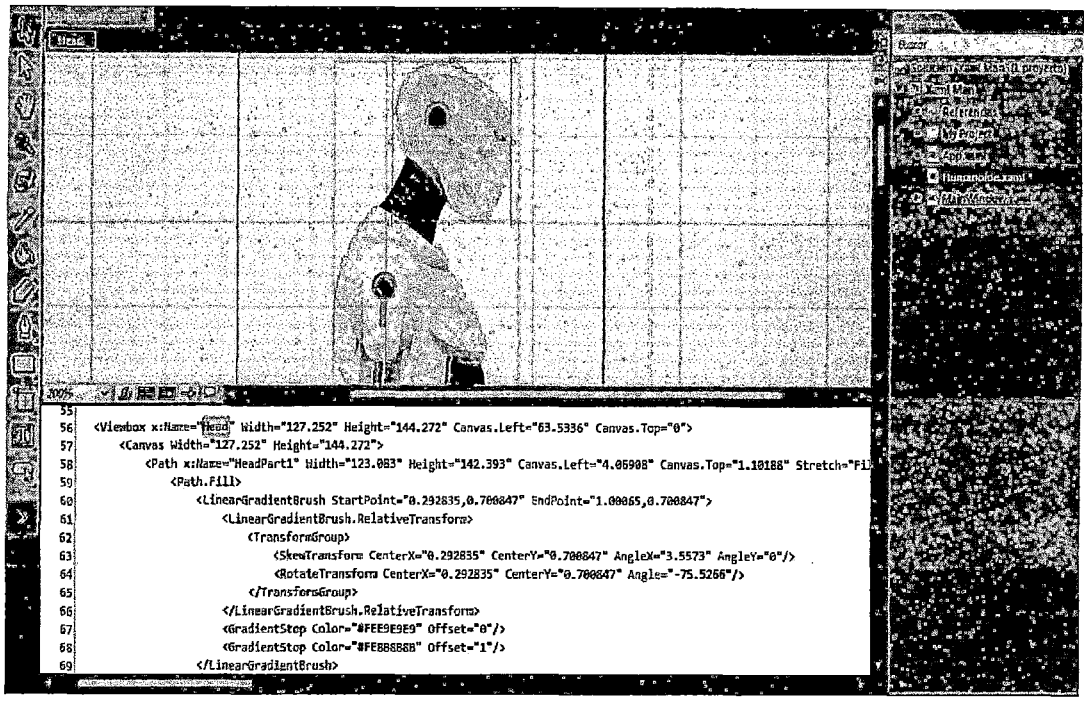


Figura 38: Diseño de un Androide, usando las herramientas que provee Microsoft Expression Blend.
Fuente: http://www.codeproject.com/Articles/142249/XAML-Man
Acceso: 25 de Marzo del 2013

2.2.2.9.5 ANIMAR EN MICROSOFT EXPRESSION BLEND

Gracias a Expression Blend las animaciones son muy sencillas de realizar y a cambio otorgan a nuestra aplicación mucho dinamismo, sin disminución de la performance.

En Microsoft Expression Blend, las animaciones se basan en fotogramas clave que definen los puntos iniciales y finales de una transición visual suave. Para crear una animación en Expression Blend, se debe crear un guión gráfico y en él se debe establecer los fotogramas clave en una escala de tiempo para marcar los cambios en las propiedades de un objeto.

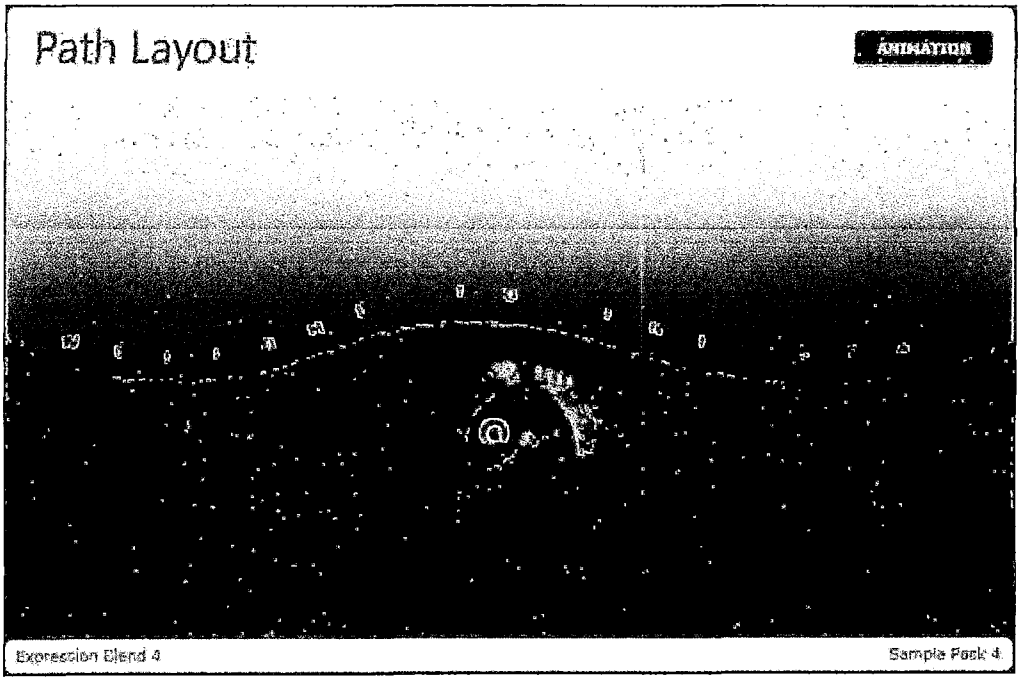


Figura 39: Ejemplo de un PathAnimation (Animacion Siguiendo Un Trazado o Ruta) en la cual se muestra la animación de un pez nadando sobre las olas (Ruta), el contorno de las olas se convierte en la guía del movimiento para el pez.

Fuente: Paquete de ejemplos incluidos en Microsoft Expression Blend.

CAPÍTULO

3

ANÁLISIS

En el presente capítulo, realizaremos una actividad conocida como toma de requerimientos, durante la cual se detallaran al máximo los requerimientos parallevar el Proyecto a buen puerto.

3.1 IDENTIFICACIÓN DE VARIABLES INDUSTRIALES

En todo proceso tenemos diversas variables, las cuales afectan las entradas o salidas de éste, por tanto estas variables deben mantenerse bien en un valor deseado fijo, bien en un valor variable con el tiempo de acuerdo con una relación predeterminada, o bien guardando una relación determinada con otra variable.

La medición de una o más variables, hace posible determinar con certeza que sucede en un punto específico de un proceso industrial, por tanto identificarlas y clasificarlas hace que su tratamiento sea más fácil. A continuación se presenta una lista que intenta reunir todas las variables existentes:

VARIABLES ELECTRICAS-MAGNETICAS		
NOMBRE	**SIMBOLO**	**UNIDAD**
AMPLITUD DE ONDA	-	-
FRECUENCIA	-	-
FLUJO (ELECTRICO,MAGNETICO)	F	Wb
CAMPO (ELECTRICO,MAGNETICO)	E,B	V/m,T
CARGA	C	CULOMBIO
ENERGIA	WxH	VATIO POR HORA
IMPEDANCIA	-	-
INDUCTANCIA	-	-
INTENSIDAD DE CORRIENTE	I	AMPERIOS
PERMEABILIDAD	Micro	H/m, N/A2
POTENCIA	W	VATIOS
RESISTENCIA	Ω	OHMIOS
TENSIÓN O VOLTAJE	V	VOLTIOS

Tabla 4: Cuadro de variables eléctricas y magnéticas

Fuente: http://www.wikipedia.com

Acceso: 25 de Marzo del 2013

VARIABLES FISICAS ESCALARES		
NOMBRE	**SIMBOLO**	**UNIDAD**
DENSIDAD	Δ	Masa/Volumen
VOLUMEN	V	Metro Cubico
LONGITUD	L	Metro
MASA	M	Kilogramo
TIEMPO	T	Segundo
FRECUENCIA ANGULAR	Θ	1/s,rad/s
FRECUENCIA	F	Hertz
POTENCIA	P	W
ENERGIA	E	J
PRESION	P	-
TENSION SUPERFICIAL	-	-
TRABAJO	W	-

Tabla 5: Cuadro de variables escalares.
Fuente: http://www.wikipedia.com
Acceso: 25 de Marzo del 2013

VARIABLES FISICAS VECTORIALES		
NOMBRE	**SIMBOLO**	**UNIDAD**
ACELERACION	A	m/sxs
RAPIDEZ	v	e/t
TRAYECTORIA	-	-
DESPLAZAMIENTO	L	M
DISTANCIA	d	M
FLUJO MAGNETICO	**F**	Wb
FUERZA	F	N
PRESION	p	Pa
MOMENTO(LINEAL , ANGULAR)	**P, L**	Kg.m/s-Js
VELOCIDAD (LINEAL , ANGULAR)	V ω	d/t rad/s

Tabla 6: Cuadro de variables vectoriales.
Fuente: http://www.wikipedia.com
Acceso: 25 de Marzo del 2013

VARIABLES CLIMATICAS-ATMOSFERICAS		
NOMBRE	**SIMBOLO**	**UNIDAD**
FLUJO	-	-
CAUDAL	-	Metro Cubico Por Segundo
HUMEDAD	H	Gramos/m cubico
PRESION	-	-
RADIACION SOLAR	RS	Irradiancia = E/s.m2
TEMPERATURA	Θ	Kelvin
TURBULENCIA	-	-

Tabla 7: Cuadro de variables climáticas y atmosféricas.
Fuente: http://www.wikipedia.com
Acceso: 25 de Marzo del 2013.

VARIABLES QUIMICAS		
NOMBRE	**SIMBOLO**	**UNIDAD**
CONDUCTIVIDAD	-	-
POTENCIAL DE HIDROGENO	PH	unidades
POTENCIAL DE OXIDACION REDUCCION	REDOX	unidades
CONCENTRACION DE GASES		Miligramos/m cubico
MOLARIDAD	M	Moles/litro
TURBIDEZ	T	NTU (Unidades Nefelometricas de Turbidez)

Tabla 8: Cuadro de variables químicas.
Fuente: http://www.wikipedia.com
Acceso: 25 de Marzo del 2013.

De los cuadros anteriores podemos concluir que las variables más comunes que se miden en los procesos industriales, tomando como referencia libros sobre instrumentación industrial, son:

Variables Físicas	Descripción
Tensión (Voltaje)	Es la presión que ejerce una fuente de suministro de energía eléctrica, sobre las cargas eléctricas en un circuito eléctrico cerrado, para que se establezca el flujo de una corriente eléctrica.
Caudal	Es el volumen de un fluido que circula a una determinada velocidad, en una unidad de tiempo.
Presión	Mide la fuerza por unidad de superficie, y sirve para caracterizar como se aplica una determinada fuerza resultante sobre una superficie.
Temperatura	Es una magnitud escalar que nos presenta cierta información sobre la energía interna contenida en un cuerpo o sustancia debido a la excitación de sus partículas.
Nivel ▪ De líquidos ▪ De sólidos.	En su sentido más general nivel hace referencia a una "altura" relativa a otra altura; generalmente se toma como punto de referencia una base. Los medidores de nivel de líquidos trabajan midiendo directamente la altura del líquido sobre una línea de referencia, mediante la presión hidrostática, el desplazamiento producido por el líquido contenido en un tanque o bien por las características eléctricas del líquido Los medidores de nivel solido son diseñados para detener el llenado de silos y diversas zonas de almacenamiento, tales como: depósitos de granos, cereales, portuarios y aeroportuarios, etc.
Velocidad	Es una magnitud física de carácter vectorial que expresa la distancia recorrida por un objeto por unidad de tiempo.
Peso	Es una medida de la fuerza gravitatoria que actúa sobre un objeto
Humedad	Es la cantidad de vapor de agua presente en el aire.
Densidad	Es una medida para determinar la cantidad de masa contenida en un determinado volumen de una sustancia.
Viscosidad	Es la resistencia de un fluido a cambiar de forma o a moverse (por la causa que sea). Esa resistencia se debe a la cohesión de las partículas que lo componen, que ejercen una especie de fricción interna que perturba el movimiento o cambio de forma

Tabla 9: Cuadro resumen de variables Físicas más usuales en la industria.

Fuente: Instrumentación Industrial – Antonio Creus Sole-Pagina 300-ed Marcombo

Fuente: Instrumentación Industrial – Alexander Espinosa.

Variables Químicas	Descripción
PH	Es una medida de acidez o alcalinidad de una disolución. El *pH* indica la concentración de iones hidronio [H3O+] presentes en determinadas sustancias.
Conductividad eléctrica	La conductividad es la capacidad de una solución para conducir una corriente eléctrica. El agua destilada pura no conduce en principio la corriente, pero si se disuelven sólidos minerales, aumenta su capacidad de conducción.
Redox	Son las reacciones químicas de transferencia de electrones, que se produce entre un conjunto de especies químicas, uno oxidante y uno reductor; provocando un cambio en sus estados de oxidación.
Nivel de gases	En la industria interesa determinar la concentración de los gases tales como CO2, CO + H2, O2, u otros, bien en el análisis de humos de salida de las calderas de vapor, para comprobar su combustión correcta, o bien en el análisis de concentración de gases desde el punto de vista de seguridad ante una eventual explosión.

Tabla 10: Cuadro resumen de variables Químicas más usuales en la industria.
Fuente: El ABC de la instrumentación Industrial en el control de procesos Industriales- Gilberto Enríqyez Harper – Editorial Limusa.

Todas las variables que intervienen en un proceso industrial, son monitoreadas y controladas por medio de la instrumentación del proceso.

Las variables restantes forman parte de procesos industriales en una escala mínima por tanto la implementación de un control que interactúe con estos, quedaran fuera del alcance de este trabajo.

3.2 HERRAMIENTAS DE DESARROLLO EMPLEADAS

I. Se hará uso de la herramienta WPF (Windows Presentation Fundation) que comprende el lenguaje de programación Microsoft Visual C# para la parte de la implementación y XAML para la parte de la presentación.

WPF es la herramienta de nuestra elección porque cuenta con muchísimos recursos que facilitan la escritura de código fuente y reducen el tiempo necesario para lograrlo, también cuenta con un elevado número de controles en la barra de herramientas, que ayudan al rápido desarrollo de aplicaciones.

Es preciso indicar que los controles con los que cuenta esta herramienta son mucho más enriquecidos en cuanto a propiedades, métodos, eventos, animaciones, entre otros, que hacen que la labor de personalización y construcción de componentes sea una tarea muy sencilla; sin duda una característica imprescindible para hacerle frente a desarrollos como se pretende en este proyecto.

Cabe resaltar también que esta herramienta no es una moda, sin duda es el presente y futuro para el desarrollo de aplicaciones en el sistema operativo Windows, puesto que Microsoft anuncio hace ya un buen tiempo que todos los desarrollos e investigaciones para su anterior plataforma de desarrollo Windows Forms han sido abandonados y se dedicará tiempo y presupuesto a ésta nueva tecnología, que pretende estar en el mercado mucho tiempo.

Además existe muchísima documentación en línea, foros de discusión especializados y canales de noticia dedicados a ésta tecnología. A pesar de tener una curva de aprendizaje muy elevada, debido a conceptos y herramientas muy novedosas y complejas de entender a primera instancia, los beneficios que se obtienen luego de esto también son muy elevados y satisfactorios, debido a esto numerosas empresas software de fama mundial, dedicados a diferentes rubros están migrando o reescribiendo sus productos para la interacción con esta herramienta.

II. Se hará uso de la herramienta de diseño de interfaz de usuario Microsoft Expression Blend, para la parte de diseño de algunos componentes, puesto que nos permite controlar la eficacia del XAML con mayor facilidad.

Con esta herramienta generaremos la interfaz para algunos controles, dibujando sus formas y contenido, aplicando estilos para lograr una apariencia totalmente profesional y proporcionar experiencias de usuario atractivas, para luego con el

Visual Studio hacer que las distintas áreas de éste, respondan a clics del mouse y a otras acciones del usuario.

Cabe resaltar que esta herramienta nos permite abrir y trabajar con proyectos de Visual Studio ya que mantiene sincronizado los cambios en los archivos, es decir al trabajar en un proyecto en Expression Blend y Visual Studio al mismo tiempo; cualquier cambio que se haga en Visual Studio se aplicará inmediatamente en Expression Blend y viceversa.

Nota:
Antes de la llegada de WPF, GDI+ de Microsoft Y Java Graphics de Oracle eran las opciones más extendidas para la construcción y redefinición de componentes, pero no por ello las más óptimas ni evolucionadas. Cabe resaltar que éstas alternativas de desarrollo aún se encuentran vigentes hoy en día pero no serán tomadas en cuenta en este trabajo.

Nota:
Tras analizar las características más importantes de este nuevo lenguaje, podemos afirmar que encontramos en él una solución global, de propósito general y accesible para cualquier programador. Además su integración con Expression Studio hace más fácil que nunca la construcción de todo tipo de aplicaciones.

Las comparaciones para la búsqueda del mejor rendimiento entre las herramientas de programación existentes en el mercado quedarán al margen, sin que ello menoscabe las virtudes y atributos que éstos poseen.

3.3 IDENTIFICACIÓN DE CLASES

El desarrollo del presente trabajo, comprende las siguientes clases:

- Componente Barras de Progreso.
- Componente Clima.
- Componente Comunicación.
- Componente Contador.
- Componente Dial.
- Componente Display.
- Componente Interruptor.
- Componente Ploteador de Señales.
- Componente Termómetro.
- Componente Velocímetro.

A continuación se dará una descripción más detallada de cada una de ellas, mediante el uso de tarjetas CRC:

CLASE: COMPONENTE BARRAS DE PROGRESO	
DESCRIPCION: Este componente es usado para representar de manera gráfica la evolución de un determinado proceso.	
RESPONSABILIDADES	**COLABORACIONES**
• Adquirir datos. • Establecer valor máximo. • Establecer valor mínimo. • Determinar cantidad de divisiones. • Indicar valor de la energía de forma gráfica. • Comparar con valor indicador. • Indicar alarma visual. • Controlar eventualidad.	• Comunicación. • Comunicación

CLASE: COMPONENTE CLIMA	
DESCRIPCION: Componente usado para mostrar el estado del tiempo en un determinado momento del día.	
RESPONSABILIDADES	**COLABORACIONES**
• Adquirir datos. • Mostrar estado de tiempo.	• Comunicación.

CLASE: COMPONENTE COMUNICACIÓN	
DESCRIPCIÓN: Componente diseñado para interactuar con datos provenientes de sensores de temperatura, presión, humedad, nivel, etc., representándolos gráficamente en los demás componentes diseñados.	
RESPONSABILIDADES	**COLABORACIONES**
• Abrir comunicación. • Enviar datos.	• Componente Barras de Progreso, Clima, Contador, Dial, Display, Interruptor, Ploteador de Señales, Termómetro y Velocímetro. • Componente Barras de Progreso,

• Recibir datos. • Cerrar comunicación	Clima, Contador, Dial, Display, Interruptor, Ploteador de Señales, Termómetro y Velocímetro. • Componente Barras de Progreso, Clima, Contador, Dial, Display, Interruptor, Ploteador de Señales, Termómetro y Velocímetro. • Componente Barras de Progreso, Clima, Contador, Dial, Display, Interruptor, Ploteador de Señales, Termómetro y Velocímetro.

CLASE: COMPONENTE CONTADOR	
DESCRIPCION: Componente diseñado para llevar la cuenta de la evolución de una variable, mientras este se encuentre en ejecución.	
RESPONSABILIDADES	**COLABORACIONES**
• Formato de adquisición de datos. • Indicar el valor actual. • Establecer valor máximo. • Establecer valor mínimo. • Determinar cantidad de dígitos.	• Comunicación.

CLASE: COMPONENTE DIAL	
DESCRIPCION: Componente diseñado para aumentar o disminuir el valor de una variable, al mover la manija o perilla en uno u otro sentido del dial.	
RESPONSABILIDADES	**COLABORACIONES**
• Formato de adquisición de datos. • Indicar el valor de la variable. • Establecer valor máximo. • Establecer valor mínima. • Determinar cantidad de divisiones. • Determinar la orientación del dial.	• Comunicación.

CLASE: COMPONENTE DISPLAY	
DESCRIPCION: Componente diseñado para visualizar el valor de una variable en forma de cantidades numéricas o cadenas de texto.	
RESPONSABILIDADES	**COLABORACIONES**
• Formato de adquisición de datos. • Indicar el valor de la velocidad. • Establecer valor de velocidad máxima. • Establecer valor de velocidad mínima. • Determinar cantidad de divisiones	• Comunicación.

CLASE: COMPONENTE INTERRUPTOR	
DESCRIPCION: Componente diseñado para prender, apagar o reiniciar algún tipo de proceso.	
RESPONSABILIDADES	**COLABORACIONES**
• Iniciar proceso. • Detener proceso. • Reiniciar Proceso	• Comunicación. • Comunicación. • Comunicación.

CLASE: COMPONENTE PLOTEADOR DE SEÑALES	
DESCRIPCION: Componente diseñado para representar la evolución de una o más variables simultáneamente, con respecto al tiempo u otro parámetro.	
RESPONSABILIDADES	**COLABORACIONES**
• Formato de adquisición de datos. • Establecer el rango de los valores de entrada. • Ubicar el origen de coordenadas. • Ajustar la escala del ploteador. • Escalar señales. • Graficar señales.	• Comunicación.

CLASE: COMPONENTE TERMÓMETRO	
DESCRIPCION: Componente usado para medir la temperatura.	
RESPONSABILIDADES	**COLABORACIONES**
• Adquirir datos. • Establecer valor máximo. • Establecer valor mínimo. • Determinar cantidad de divisiones. • Indicar valor de temperatura. • Comparar con valor indicador. • Indicar alarma visual. • Controlar eventualidad.	• Comunicación. • Comunicación.

CLASE: COMPONENTE VELOCÍMETRO	
DESCRIPCION: Componente diseñado para medir la velocidad u otra magnitud, como la presión de gases, resistencia, voltaje, amperímetro, etc.	
RESPONSABILIDADES	**COLABORACIONES**
• Adquirir datos. • Establecer valor de velocidad máxima. • Establecer valor de velocidad mínima. • Determinar cantidad de divisiones • Indicar el valor de la velocidad.	• Comunicación.

CAPÍTULO

4

DISEÑO

En el presente capítulo modelaremos al sistema y encontraremos su forma, incluidasu arquitectura.Se explica además cómo se conseguirá las funcionalidades deseadas, por ejemplo el tipo y la cantidad de clases a emplear.

4.1 DISEÑO COMPONENTE BARRAS DE PROGRESO

4.1.1 DESCRIPCIÓN

El Componente Barras de progreso está diseñado para representar de forma gráfica, la evolución de un determinado proceso.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricciones**
ValorActual	Doublé	Valor numérico que recibe el componente	>0
NumeroProgeso	Int	Representa de manera numérica el valor de la propiedad ValorActual.	>0
gradiente_1	Color	Uno de los colores que representa el progreso de la energía.	¡null
gradiente_2	Color	Uno de los colores que representa el progreso de la energía.	¡null

4.1.2 MÉTODOS

Nombre: CambiarValorActual(e)
Funcionalidad:

- Coge el valor nuevo y el valor anterior.
- Valida el valor nuevo y el valor anterior con el valor del ancho del componente y que sea mayor a cero respectivamente.
- Llama al método AnimarRectangulo().

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
E	DependencyPropertyChangedEventArgs	Ninguno	Ninguno

Nombre: AnimarRectangulo(valor_anterior ,valor_nuevo)
Funcionalidad:

- Toma como referencia los valores de las variables valor_nuevo y valor_anterior.
- Realiza la animación del ancho del rectángulo que va desde el valor de la variable valor_anterior hacia el valor de la variable valor_nuevo.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Doublé	valor_nuevo	Ninguno	Ninguno
Doublé	valor_anterior		

4.2 DISEÑO COMPONENTE CLIMA

4.2.1 DESCRIPCIÓN

Se ha diseñado cinco modelos diferentes que representan cinco estados distintos del clima de un día determinado. Cada diseño sirve para representar el valor numérico del tiempo, en un momento determinado del día, de manera gráfica.

A continuación se detalla la estructura de uno de los componentes.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricciones**
ValorActualsnll	doublé	Valor numérico que recibe el componente, para representar de manera gráfica el valor de la propiedad ValorActualsnll según sea el caso.	>0

4.2.2 MÉTODOS

Nombre: EstablecerNubes()
Funcionalidad:

- Fija en una posición dada distinta de la posición original de las nubes (path).
- Oculta las nubes para ser mostradas ya cuando se instancia el componente.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

Nombre: AnimarNubes()
Funcionalidad:

- Llama al método Animar para cada nube con las coordenadas originales de cada una de ellas.
- Llama al método AnimarGotas().

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

Nombre: Animar (nube, newX, newY)
Funcionalidad:

- Realiza la animación para la propiedad de la traslación de cada nube.
- La animación se realiza desde la posición actual hacia la posición original de cada nube.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Nube	Viewbox	Ninguno	Ninguno
newX	Doublé		
newY	Doublé		

Nombre: AnimarGotas()
Funcionalidad:

- Crea números aleatorios para representar tiempos distintos.
- Crea un timer que llamara a su evento timer_Click según el tiempo generado en los números aleatorios.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

Nombre: Mover (Gota, newX, newY)
Funcionalidad:

- Hace visible la gota que es un path que representara la lluvia.
- Realiza la animación de la gota que va desde su posición original hacía la nueva posición dada en los valores de las variables newX y newY.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Gota	Path	Ninguno	Ninguno
newX	doublé		
newY	doublé		

4.2.3 EVENTOS

Nombre: Mover (Gota, newX, newY)
Funcionalidad:

- Valida la cantidad de gotas que se tiene.
- Llama al método Mover() para realizar la animación de las gotas.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

4.3 DISEÑO COMPONENTE COMUNICACIÓN

4.3.1 DESCRIPCIÓN

El componente comunicación está diseñado para poder obtener datos provenientes de sensores de temperatura, presión, etc. usando el puerto serie como medio de transmisión de éstos, hacia los componentes diseñados para poder representarlos gráficamente.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricciones**
TasaDeBaudios	String	Contiene el valor de la velocidad en baudios del puerto serie.	!=null
Paridad	String	Contiene el valor del protocolo de comprobación de la paridad.	!=null
BitsDeParada	String	Contiene el valor del número estándar de bits de parada por byte.	!=null
BitsDeDatos	String	Contiene el valor de la longitud estándar de los bits de datos por byte.	!=null
NombrePuerto	String	Contiene el nombre del puerto de comunicaciones, incluido por lo menos todos los puertos COM disponibles.	!=null
TipoTransmision	enum	Contiene el tipo de transmisión de datos.	Text o Hex

4.3.2 MÉTODOS

Nombre: CargarNombrePuertos(obj)
Funcionalidad:

- Obtiene el nombre de todos los puertos serie disponibles en la PC.
- Llena un ComboBox disponible para visualizar los nombres de los puertos serie.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
obj	object	Ninguno	Ninguno

Nombre: HexadecimalABytes(msg)
Funcionalidad:

- Obtiene el valor leído por el puerto serie como parámetro de entrada.
- Convierte el valor obtenido a un arreglo de byte.
- Devuelve como valor de salida el arreglo de byte.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
msg	String	HexadecimalABytes	byte[]

Nombre: DisplayData(DatosPuertoSerie)
Funcionalidad:

- Obtiene como parámetro de entrada los datos leídos del puerto serie en el tipo de transmisión Text.
- Recorre cada carácter del parámetro de entrada para ubicar los delimitadores de cada dato como el delimitador de finalización de envió de datos.
- Cada dato es almacenado en un variable Dato1, Dato2, etc.
- Se asigna cada Dato1 a uno de los componentes diseñados.

Parámetros de Entrada		**Parámetros de Salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
DatosPuertoSerie	string	Ninguno	Ninguno

Nombre: AbrirPuerto()
Funcionalidad:

- Realiza la apertura del puerto COM seleccionado.
- Asigna el valor correspondiente a cada atributo creado en este componente (Paridad, NombrePuerto, ect.).

Parámetros de Entrada		**Parámetros de Salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

4.3.3 EVENTOS

Nombre: PuertoCom_DataReceived(sender ,e)
Funcionalidad:

- Evento que hace la recepción de datos desde el puerto serie.
- Selecciona el tipo de transmisión Text o Hex.
- Lee datos desde el puerto serie.
- Llama al evento DisplayData.

Parámetros de Entrada		**Parámetros de Salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
sender	Object	Ninguno	Ninguno
e	SerialDataReceivedEventArgs		

4.4 DISEÑO COMPONENTE CONTADOR

4.4.1 DESCRIPCIÓN

El Componente Contador está diseñado para llevar la cuenta de la evolución de una variable, mientras éste se encuentre en ejecución. Para que esto ocurra, se fija el valor inicial a partir del cual empezara la cuenta y un valor final, que indica que la cuenta debe llegar a ese límite y luego detenerse.

Para representar las cantidades, se establece también el número de dígitos con los cuales se representara una determinada cantidad, por ejemplo el número tres lo podemos representar como "00003", "03", etc. Dependiendo si se estableció cinco o dos dígitos respectivamente, para representar esta cantidad.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricción**
NroInicial	Double	Valor de partida.	>=0
NroFinal	Double	Valor final o de llegada.	>=0
NroResaltado	Double	Valor que se muestra inicialmente al cargar el control.	>=0
CantidadDigitos	Double	Numero de dígitos usados para representar una cantidad.	>0

4.4.2 MÉTODOS

Nombre: EstablecerNroAlCargar()
Funcionalidad: De los nueve dígitos que contienen nuestro sistema de numeración, indica cuál de ellos debe estar resaltado o visible al inicio.

Parámetros de Entrada		**Parámetros de Salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

Nombre: AnimacionMostrarNuevoNumero (cnv, LimiteSuperior, LimiteInferior)
Funcionalidad: Contiene la animación del control, que crea el efecto de desplazamiento de los dígitos.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Cnv	Canvas	Ninguno	Ninguno
LimiteSuperior	Int	Ninguno	Ninguno
LimiteInferior	Int	Ninguno	Ninguno

4.5 DISEÑO COMPONENTE DIAL

4.5.1 DESCRIPCIÓN

El Componente Dial está diseñado para aumentar o disminuir el valor de una variable, al mover la manija o perilla en uno u otro sentido del dial. Para que esto ocurra, se fija el valor inicial y final o valores límite, dentro de los cuales se dibujaran las marcas indicando un valor, además se debe indicar la orientación de los valores, ya sea en sentido horario o anti horario, si ocupara todo el circulo o una porción de este. También se debe indicar cuantas divisiones debe tener el control, entre estos valores límite.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricción**
ValorActual	Double	Valor que muestra el Dial.	>=0
AnguloInicial	Double	Angulo desde el cual empezara a graficarse los números y las marcas.	>=0
Cirunferencia	Double	Valor que indica si los números y marcas se dibujan en todo el círculo o una porción de este.	>=0
DistanciaTicks	Double	Indica la distancia a partir de la circunferencia, desde la cual se dibujaran las marcas.	>=0
DistanciaNros	Double	Indica la distancia a partir de la circunferencia, desde la cual se dibujaran los números.	>=0
LongMinTicks	Double	Indica el tamaño de las marcas pequeñas.	>=0
LongMaxTicks	Double	Indica el tamaño de las marcas grandes.	>=0
ValorMin	Double	Indica el valor mínimo con la cual se empezara la cuenta para el Dial.	>=0
ValorMax	Double	Indica el valor máximo con la cual se terminara la cuenta para el Dial.	>0
NroDivisiones	Int	Indica el número de divisiones que tendrá el dial, comprendidos entre el valor máximo y mínimo.	>0

ShowNumbers	Bool	Indica si se muestran los números para el Dial.	!null
ShowSmallScale	Bool	Indica si se muestran las marcas pequeñas para el Dial.	!null
ShowLargeScale	Bool	Indica si se muestran las marcas grandes para el Dial.	!null
LadoDial	Double	Indica el tamaño para el radio del Dial.	>0
ColNumeros	Color	Indica el color para los números del Dial.	!null
ColSmallTicks	Color	Indica el color para las marcas pequeñas.	!null
ColLargeTicks	Color	Indica el color para las marcas grandes.	!null
ColDial	Color	Indica el color para el Dial.	!null
Angulo	Double	Indica a partir de que cuadrante se empiezan a dibujar los números y marcas para el Dial.	>=0

4.5.2 MÉTODOS

Nombre: DibujarDial()
Funcionalidad: Este método se encarga de dibujar el círculo grande y ubicar los números y marcas para este.

Parámetros de entrada		**Parámetros de salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Void

Nombre: DibujarManija()
Funcionalidad: Este método se encarga de dibujar la manija o perilla, que se mueve alrededor del control.

Parámetros de entrada		**Parámetros de salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Void

Nombre: ObtenerPosisionManija(Valor)
Funcionalidad: Este método se encarga de calcular la posición de la perilla, en base a un valor pasado como parámetro. Este método es usado por DibujarManija()

Parámetros de entrada		**Parámetros de salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Valor	Double	Pos	Point

4.5.3 EVENTOS

Nombre: Dial_MouseLeftButtonDown(sender,e)
Funcionalidad: Este evento se activa cuando presionamos el botón izquierdo del mouse para intentar mover la manija o perilla del Dial, luego captura la posición inicial del mouse al momento de presionar la perilla.

Parámetros de entrada		**Parámetros de salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Sender	Object	Ninguno	Void
e	MouseButtonEventArgs		

Nombre: Dial_MouseLeftButtonUp(sender,e)
Funcionalidad: Este evento se activa cuando soltamos el botón izquierdo del mouse luego de mover la manija o perilla del Dial, luego captura la posición final del mouse al momento de soltar el ratón.

Parámetros de entrada		**Parámetros de salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Sender	Object	Ninguno	Void
E	MouseButtonEventArgs		

Nombre: Dial_MouseMove(sender,e)
Funcionalidad: Este evento se activa cuando movemos la perilla alrededor del Dial.

Parámetros de entrada		**Parámetros de salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Sender	Object	Ninguno	Void
E	MouseEventArgs		

4.6 DISEÑO COMPONENTE DISPLAY

4.6.1 DESCRIPCIÓN

El componente Display está diseñado para mostrar el valor de una variable en forma numérica o de cadenas de caracteres, o simplemente mostrar mensajes personalizados, que se van pasando por la pantalla en un determinado sentido.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricción**
ColorDisabled	Brush	Color para un pixel desactivado.	!null
ColorHabilitado	Brush	Color para un pixel encendido.	!null
DireccionMovimiento	Enum	Valor que indica en qué sentido se desplaza el mensaje.	!null
Estilo	Enum	Valor que indica si los pixeles se pintan en forma rectangular o circular.	!null
DisplayText	String	Es el mensaje que se muestra en la pantalla.	!null
LadoPixel	Double	Indica el tamaño de los pixeles.	>0
RadiusXY	Double	Indica la curvatura de los pixeles.	>=0
NroFilas	Int	Indica la cantidad de filas que se usaran para representar un digito.	>0
NroColumnas	Int	Indica la cantidad de columnas que se usaran para representar un digito.	>0
ValorMin	Double	Indica el valor mínimo con la cual se empezara la cuenta para el Dial.	>=0
CharDict	Dictionary	Estructura que se usa para almacenar un carácter y su lista de pixeles que lo conforman	!null
SimbolsDict	Dictionary	Estructura que se usa para almacenar un carácter especial y su lista de pixeles que lo conforman	!null
Empezar	Bool	Valor usado para empezar o detener la animación del mensaje en pantalla.	!null

4.6.2 MÉTODOS

Nombre: DibujarCaracter(BitsCaracter,Posicion)
Funcionalidad: Este método se encarga de dibujar un carácter cualquiera, en una determinada posición de la pantalla, en base a los pixeles que se le pasan.

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
BitsCaracter	Ulong	Ninguno	Void
Posicion	Double		

Nombre: DibujarTexto(Texto)
Funcionalidad: Este método se encarga de dibujar todos los caracteres que conforman el mensaje o parámetro de entrada. Para eso hace uso del método DibujarCaracter(BitsCaracter,Posicion).

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Texto	String	Ninguno	Void

Nombre: DiccionarioDeCaracteres()
Funcionalidad: Este método se encarga de llenar una estructura de tipo Clave-Valor, en la cual se almacena un carácter especial y su respectiva lista de pixeles que lo representa.

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Void

Nombre: DiccionarioDeSimbolos()
Funcionalidad: Este método se encarga de llenar una estructura de tipo Clave-Valor, en la cual se almacena un carácter conocido y su respectiva lista de pixeles que lo representa.

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Void

Nombre: EmpezarAnimacion()
Funcionalidad: Este método se encarga de animar el mensaje en la pantalla, en una determinada dirección.

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Void

4.7 DISEÑO COMPONENTE INTERRUPTOR

4.7.1 DESCRIPCIÓN

El Componente Interruptor está diseñado para prender, apagar, detener o reiniciar algún tipo de proceso.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricción**
IsChecked	Bool	Indica si el estado del interruptor está activado o desactivado.	!null
PosActual	Enum	Valor que indica si el interruptor apunta hacia arriba o abajo del control.	!null

4.7.2 MÉTODOS

Nombre: CambiarEstado()
Funcionalidad: Este método se encarga de cambiar el estado del control, es decir rota la manija hacia arriba o abajo, para indicar si se encuentra activo o desactivo.

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Void

4.7.3 EVENTOS

Nombre: MyJoystick_MouseLeftButtonDown(sender,e)
Funcionalidad: Este evento se activa cuando presionamos el botón izquierdo del mouse para intentar mover la manija o perilla del control, luego calcula si la posición

actual hacia donde apunta la manija es hacia arriba o abaja, para luego realizar una rotación vertical de la manija.

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Sender	Object	Ninguno	Void
E	MouseButtonEventArgs		

Nombre: MyJoystick _MouseLeftButtonUp(sender,e)
Funcionalidad: Este evento se activa cuando soltamos el botón izquierdo del mouse luego de mover la manija o perilla del Dial, para luego liberar la captura del mouse.

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Sender	Object	Ninguno	Void
E	MouseButtonEventArgs		

Nombre: OnCheckedChanged (sender,e)
Funcionalidad: Este evento se activa cuando movemos la perilla alrededor del Dial.

Parámetros de entrada		Parámetros de salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Sender	Object	Ninguno	Void
E	DependencyPropertyChangedEventArgs		

4.8 DISEÑO COMPONENTE PLOTEADOR DE SEÑALES

4.8.1 DESCRIPCIÓN

El Componente Ploteador de Señales está diseñado para representar la evolución de una o más variables simultáneamente, con respecto al tiempo u otro parámetro.

Este componente además es capaz de graficar y también animar cualquier tipo de función matemática, un arreglo de puntos X, Y o un arreglo de puntos lineal, con respeto al tiempo.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricción**
CanvasAreaDeDibujo	Canvas	Superficie sobre la cual se dibujaran y animaran los distintos tipos de funciones.	!null
XLabel	String	Leyenda que se muestra en el eje X.	-
YLabel	String	Leyenda que se muestra en el eje Y.	-
XLabelColor	Color	Color para representar la leyenda en el eje X.	!null
YLabelColor	Color	Color para representar la leyenda en el eje Y.	!null
ColorFondo	Color	Color usado para el fondo del control.	!null
EstiloLinea	Enum	Indica el tipo de línea a usar para dibujar las grillas del control.	!null
OrigenDeCoodenadas	Enum	Indica el lugar donde se ubicara el origen de coordenadas.	!null
EscalaX	Double	Valor usado para hacer coincidir los puntos X, de un gráfico cualquiera, con los del ploteador.	>0
EscalaY	Double	Valor usado para hacer coincidir los puntos Y, de un gráfico cualquiera, con los del ploteador.	>0
NroDivX	Double	Representa el número de partes iguales en las que se dividirá el ancho del ploteador.	>0
NroDivY	Double	Representa el número de partes iguales en las que se dividirá el alto del ploteador.	>0
GrillaX	Bool	Indica si se visualiza o no las grillas horizontales en el ploteador.	!null
GrillaY	Bool	Indica si se visualiza o no las grillas verticales en el ploteador.	!null
ColorGrilla	Color	Representa el color con el cual se dibujaran las grillas del control.	!null
ColorNumeros	Color	Representa el color con el cual se dibujaran los números del control.	!null

4.8.2 MÉTODOS

Nombre: XYLabels()
Funcionalidad: Este método se encarga de dibujar las leyendas para los ejes X e Y.

Parámetros de entrada		**Parámetros de salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Void

Nombre: DibujarEjes(MyCanvas)
Funcionalidad: Este método se encarga de dibujar los ejes X e Y, en una ubicación determinada, también se encarga de dibujar las grillas para el control, dibujar los números y calcular la escala para el mismo.

Parámetros de entrada		**Parámetros de salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
MyCanvas	Canvas	Ninguno	Void

4.9 DISEÑO COMPONENTE TERMÓMETRO

4.9.1 DESCRIPCIÓN

El Componente Termómetro está diseñado para mostrar el valor de una temperatura de manera gráfica, fijando un valor indicador entre los valores límites de la escala, que podrá representar alguna ocurrencia fuera de lo normal en algún proceso si el valor de la temperatura supera este valor indicador establecido. Para lo cual ocurrirá una alarma de manera visual (cambiando el color del contenido del componente termómetro) que podrá alertar al operario de cualquier eventualidad.

Atributos			
Nombre	**Tipo**	**Descripcion**	**Restricción**
ValorActual	Doublé	Valor que marca el componente en un momento dado.	<ValorMaximo >ValorMinimo
EscalaBarrido	Doublé	Valor ajustado al tamaño del Componente.	>0
CantidadDivisionesMayores	Doublé	Cantidad de marcas mayores que se graficara en la escala.	>0
CantidadDivisionesMenores	Doublé	Cantidad de marcas menores que se graficara en la escala.	>0
ValorMaximo	Doublé	Valor máximo que puede tomar el valor actual.	>ValorMinimo
ValorMinimo	Doublé	Valor mínimo que puede tomar el valor actual.	<ValorMaximo
TamañoMarcasMayores	Size	Tamaño de los rectángulos que representan las marcas mayores.	>0
ColorMarcasMayores	Color	Color usado para representar las marcas mayores de la escala.	¡null
EscalaInicialX	Doublé	Posición en eje x de la escala con origen en la parte inferior	>=0

		izquierdo del componente.	
EscalaInicialY	Doublé	Posición en eje y de la escala con origen en la parte inferior izquierdo del componente.	>=0
EscalaTextoX	Doublé	Posición en eje x de los numerales de la escala con origen en la parte inferior izquierdo del componente.	>=0
EsaclaTextoY	Doublé	Posición en eje y de los numerales de la escala con origen en la parte inferior izquierdo del componente.	>=0
EscalaTextoFuenteTamaño	Doublé	Tamaño ajustado a la fuente de los números de la escala.	>0
EscalaTamañoTexto	Size	Tamaño del recuadro donde se ponen los numerales	>0
EscalaTextoColor	Color	Color usado para representar los numerales de la escala.	¡null
TamañoMarcasMenores	Size	Tamaño de los rectángulos que representan las marcas menores.	>0
ColorMarcasMenores	Color	Color usado para representar las marcas menores de la escala.	¡null
ValorPresicionEscala	Int	Para que el valor actual sea más preciso al valor en la escala.	>0

4.9.2 MÉTODOS

Nombre: DibujarEscala()
Funcionalidad:

- Calcula el valor para el espacio entre marcas mayores.
- Grafica el tamaño de las marcas mayores.
- Calcula y grafica los números correspondientes a cada marca mayor.
- Calcula el valor para el espacio entre marcas menores.
- Grafica el tamaño de las marcas menores.

Parámetros de Entrada		**Parámetros de Salida**	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

Nombre: CambiarValorActual(e)
Funcionalidad:

- Coge el valor nuevo y el valor anterior.
- Valida el valor nuevo y el valor anterior con los valores máximo y mínimo de la escala.

- Calcula el valor coeficiente para sincronizar el valor actual con el valor de la escala.
- Asigna el valor nuevo y el valor anterior a dos variables globales.
- Llama al método AnimarRectangulo().

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
E	DependencyPropertyChangedEventArgs	Ninguno	Ninguno

Nombre: AnimarRectangulo()
Funcionalidad:

- Compara el valor actual con el valor fijador o límite.
- Realiza la animación del componente que simula el líquido del termómetro.
- Cambia el color de degradado del líquido del termómetro según sea el caso.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

Nombre: IniciarDegradados()
Funcionalidad:

- Crea los colores de degradado lineal para el líquido del termómetro.
- Crea los colores de degradado radial para la base del termómetro.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

4.9.3 EVENTOS

Nombre: OnCurrentValuePropertyChanged(d,e)
Funcionalidad:

- Se lanza cada vez que cambia la Propiedad de Dependencia ValorActual.
- Llama al método CambiarValorActual(e).

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
D	DependencyObject	Ninguno	Ninguno
E	DependecyPropertyChangedEventArgs		

Nombre: OnPropertyChange(sender,e)
Funcionalidad:

- Se lanza cada vez que se cambia algunos de los valores de las Propiedades de Dependencia menos el valor de la Propiedad de Dependencia ValorActual.
- Llama al método DibujarEsacala().

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Sender	DependencyObject	Ninguno	Ninguno
E	DependecyPropertyChangedEventArgs		

Nombre: sb_Completed(sender,e)
Funcionalidad:

- Se lanza cada vez que la animación finaliza.
- Vuelve a realizar la animación.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Sender	Object	Ninguno	Ninguno
E	EventArgs		

4.10 DISEÑO COMPONENTE VELOCÍMETRO

4.10.1 DESCRIPCIÓN

El componente velocímetro está diseñado para mostrar el valor de una velocidad de manera gráfica. Se han diseñado modelos convencionales similares a los que traen los automóviles, diseño de circunferencia completa y cuarto de circunferencia todos los modelos implementados con la misma funcionalidad.

Atributos			
Nombre	**Tipo**	**Descripción**	**Restricciones**
ValorActual	Doublé	Valor que marca el componente en un momento dado.	<ValorMaximo >ValorMinimo
EscalaBarridoAngulo	Doublé	Valor ajustado al modelo del Componente.	>0
CantidadDivisionesMayores	doublé	Cantidad de marcas mayores que se graficara en la escala.	>0
CantidadDivisionesMenores	doublé	Cantidad de marcas menores que se graficara en la escala.	>0
ValorMaximo	doublé	Valor máximo que puede tomar el valor actual.	>ValorMinimo
ValorMinimo	doublé	Valor mínimo que puede tomar el valor actual.	<ValorMaximo
TamañoMarcasMayores	Size	Tamaño de los rectángulos que representan las marcas mayores.	>0
ColorMarcasMayores	Color	Color usado para representar las marcas mayores de la escala.	¡null
RadioEscala	doublé	Distancia, del centro del componente hacia el borde del mismo, en donde se graficara la escala, adecuado al modelo.	>0
EscalaRadioTexto	doublé	Distancia, del centro del componente hacia el borde del mismo, en donde se graficara los numerales de la escala, adecuado al modelo.	>0
EscalaTextoFuenteTamaño	doublé	Tamaño ajustado a la fuente de los números de la escala.	>0
EscalaTamañoTexto	Size	Tamaño del recuadro donde se ponen los numerales	>0
EscalaTextoColor	Color	Color usado para representar los numerales de la escala.	¡null
TamañoMarcasMenores	Size	Tamaño de los rectángulos que representan las marcas menores.	>0
ColorMarcasMenores	Color	Color usado para representar las marcas menores de la escala.	¡null
ValorPresicionEscala	Int	Para que el valor actual sea más preciso al valor en la escala.	>0
AnguloInicialEscala	double	Posición de donde se quiere que empiece la escala.	>0

4.10.2 MÉTODOS

Nombre: DibujarEscala()
Funcionalidad:

- Calcula el valor del ángulo para el espacio entre marcas mayores.
- Grafica el tamaño de las marcas mayores.

- Calcula y grafica los números correspondientes a cada marca mayor.
- Calcula el valor del ángulo para el espacio entre marcas menores.
- Grafica el tamaño de las marcas menores.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

Nombre: AnimarAguja()
Funcionalidad:

- Toma como referencia los valores de dos variables, ángulo anterior y ángulo actual.
- Aplica la rotación correspondiente a la aguja.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ninguno	Ninguno

Nombre: MoverAguja(ángulo)
Funcionalidad:

- Realiza la rotación de la aguja del componente hacia el valor que contiene la variable ángulo.
- Ubica la aguja del componente en el valor inicial de la escala.

Parámetros de Entrada		Parámetros de Salida	
Nombre	**Tipo**	**Nombre**	**Tipo**
Ninguno	Ninguno	Ángulo	double

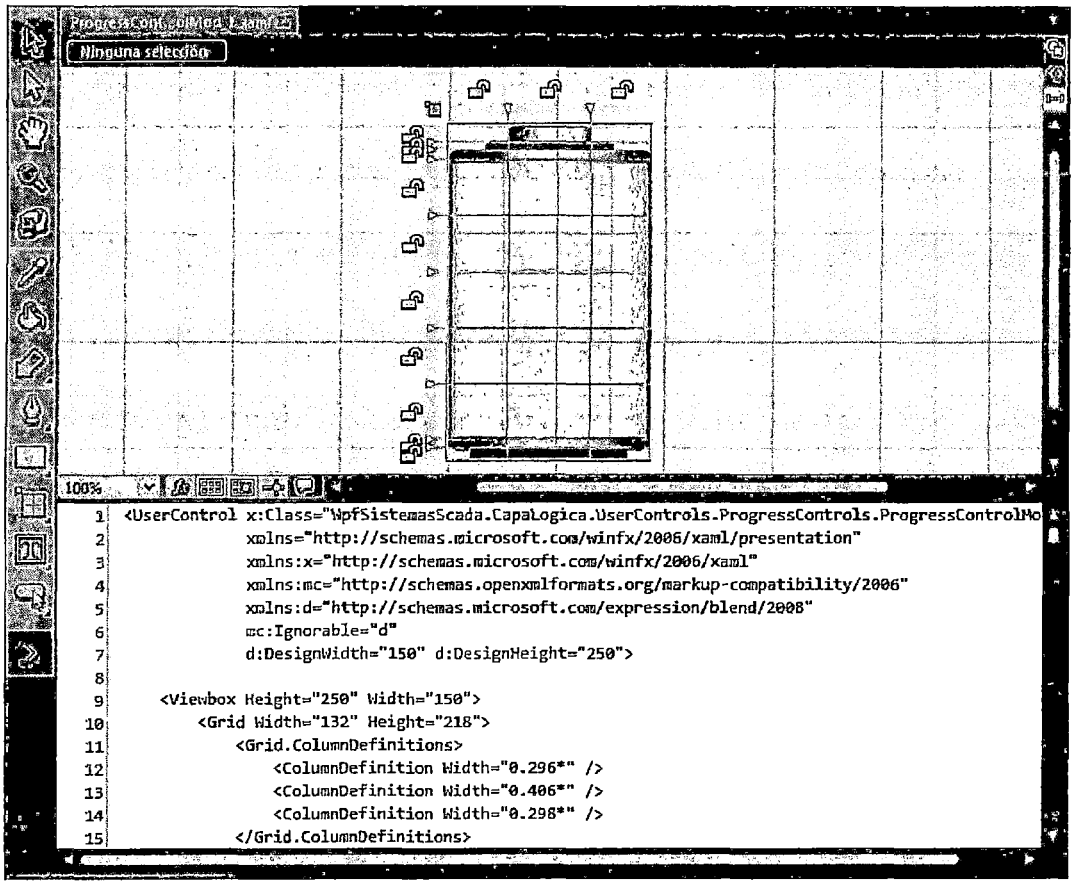
CAPÍTULO 5

IMPLEMENTACÍON

En el presente capítulo la actividad principal es la programación de las soluciones diseñadas, complementando con una labor de análisis continuada, de forma que el desarrollo vaya adaptándose a necesidades que surjan a medida que van descubriéndose dificultades, problemas imprevistos, o para ajustarse a nuevas necesidades.

5.1 IMPLEMENTACIÓN COMPONENTE BARRAS DE PROGRESO

Con la ayuda de la herramienta Expression Blend, el diseño final de este componente luce del siguiente modo:



Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

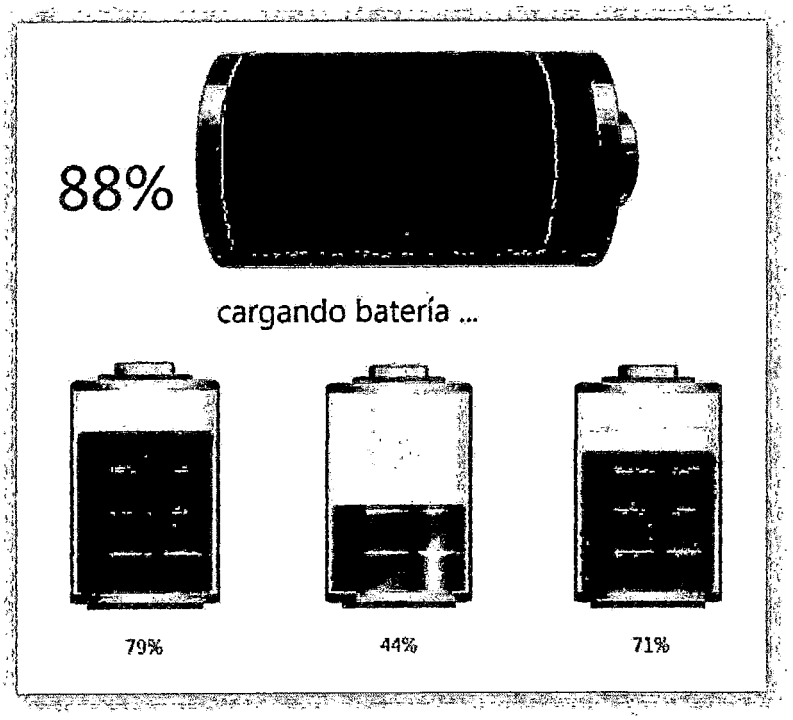
5.1.1 PROPIEDADES

```
public partial class ProgressControlMod_1 : UserControl
{
    public double ValorActual...
    public double CurrentLevel...
```

5.1.2 MÉTODOS

```
private static void CurrentTextPropertyChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)...
private static void OnCurrentValuePropertyChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)...
public  void asignarvalores(DependencyPropertyChangedEventArgs e)...
public  void animar(double valor_nuevo, double valor_anterior)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:



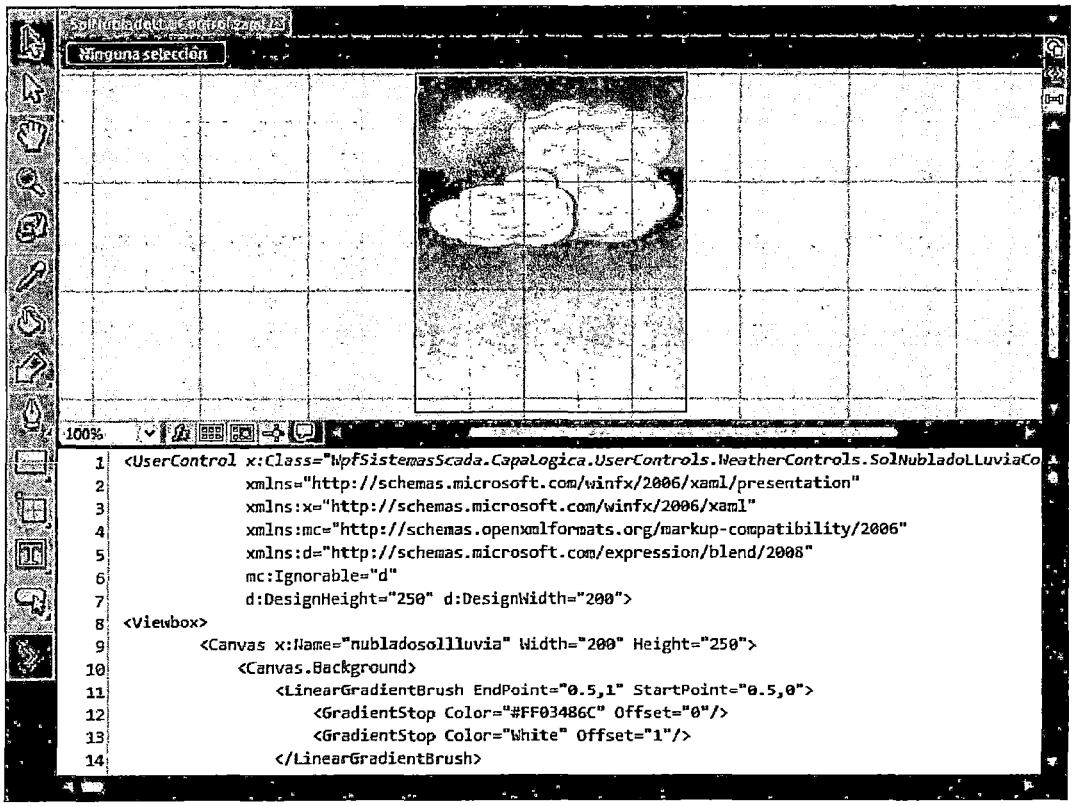
Lo más resaltante en la implementación de este componente se encuentra en el evento *OnCurrentValuePropertyChange()* y el método *animar(valor_nuevo, valor_anterior)*. A continuación una breve explicación del desarrollo de cada uno.

- *OnCurrentValuePropertyChange*(); La funcionalidad de este evento es muy importante ya que trabaja directamente con la propiedad de dependencia ValorActual. Lo que ocurre es que este evento es lanzado cada vez que el valor de esta propiedad de dependencia es modificado, por consiguiente se realiza la animación correspondiente cada vez que se lanza este evento.

- *animar (valor_nuevo, valor_anterior);* Este método lo que hace es realizar la animación de la propiedad width o height (según sea el caso) de un rectángulo que simula el progreso de cargado de una batería. Su tamaño establecido es representado como el 100% para representar el porcentaje de cargado de la batería.

5.2 IMPLEMENTACIÓN COMPONENTE CLIMA

Con la ayuda de la herramienta Expression Blend, el diseño final de este componente luce del siguiente modo:



Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

5.2.1 PROPIEDADES

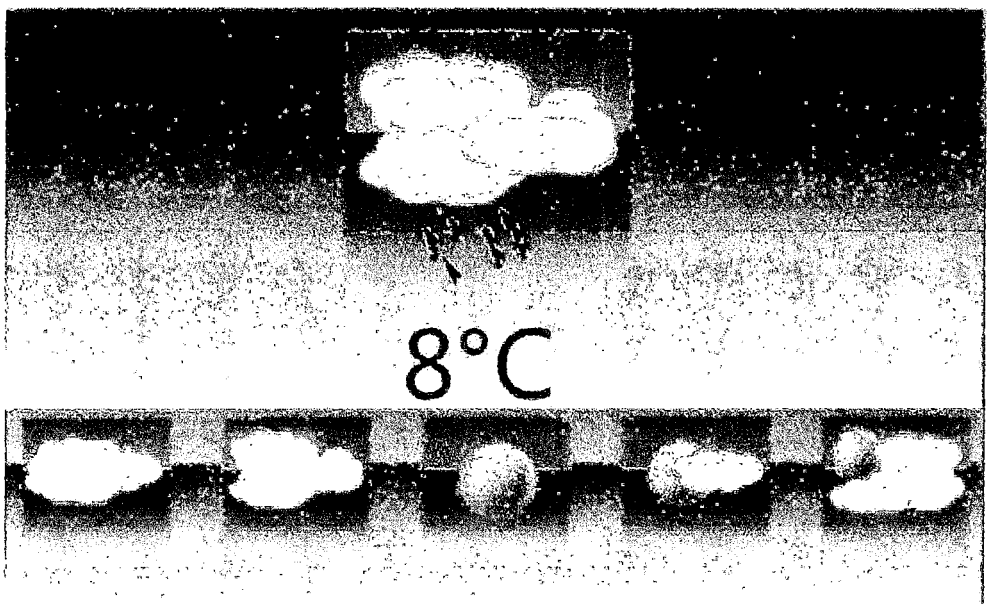
```
public partial class SolNubladoLLuviaControl : UserControl
{
    double posicion1Xoriginal;
    double posicion1Yoriginal;
    double posicion2Xoriginal;
    double posicion2Yoriginal;
    double posicion3Xoriginal;
    double posicion3Yoriginal;

    int contadorgotas;
    double posiciongotaX;
    double posiciongotaY;
    DispatcherTimer timer;
    public double ValorActualsnll...
```

5.2.2 MÉTODOS

```
private static void OnPropertyChange(DependencyObject sender, DependencyPropertyChangedEventArgs e)...
public  void  establecernubes()...
public  void  animarnubes()...
public  void  animar(Viewbox nube, double newX, double newY)...
public  void  animargotas()...
private void  timer_Tick(object sender, EventArgs e)...
public  void  mover(Path gota, double newX, double newY)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:



Lo más resaltante de la implementación de este componente se encuentra en los métodos *animar()* y *mover(gota, newX, newY).* A continuación una breve explicación del desarrollo de cada uno.

- *animar ();* Este método lo que hace es capturar las coordenadas originales de cada nube y después realizar la animación correspondiente a la propiedad traslación de cada nube
- *mover(gota, newX, newY);* Este método lo que hace es realizar la animación de una gota que se ejecuta cada vez que se inicia un timer con un tiempo menor establecido por un numero aleatorio, para darle el realismo de una lluvia. La animación se realiza igualmente a la propiedad de traslación de cada gota.

5.3 IMPLEMENTACIÓN COMPONENTE COMUNICACIÓN

Para implementar la funcionalidad de éste componente, a continuación se listarán las propiedades, métodos y eventos que se usaron:

5.3.1 PROPIEDADES

```
namespace WpfSistemasScada
{
    public class CConeccionManager
    {
        public  enum TipoDeTransmision { Text, Hex }
        private TipoDeTransmision aTipoTransmision = TipoDeTransmision.Text;
        private string aTasaDeBaudios;
        private string aParidad;
        private string aBitsDeParada;
        private string aBitsDeDatos;
        private string aNombrePuerto;
        private SerialPort aPuertoCom;
```

5.3.2 MÉTODOS

```
public  void   CargarBitsDeDatos(object obj)...
public  void   CargarBitsDeParidad(object obj)...
public  void   CargarBitsDeParada(object obj)...
public  void   CargarNombrePuertos(object obj)...
public  void   CargarTasaDeBaudios(object obj)...
private byte[] HexadecimalABytes(string msg)...
private string BytesAHexadecimal(byte[] comByte)...
private void   InicializarParametros()...
private void   AnimarGraficoSerialPort(object sender, EventArgs e)...
public  void   IniciarAnimacion()...
public  void   DisplayData(string DatosPuertoSerie)...
public  void   AbrirPuerto()...
public  void   EnviarDatos(string msg)...
```

5.3.3 EVENTOS

```
private void PuertoCom_DataReceived(object sender, SerialDataReceivedEventArgs e)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:

Lo más resaltante en la implementación de este control son los métodos *DisplayData (string DatosPuertoSerie)*, *IniciarAnimacion()* y el evento *PuertoCom_DataReceived(object sender, SerialDataReceivedEventArgs e)*, cuyos detalles más sobresalientes pasamos a describir a continuación:

- *DisplayData;* Este método se encarga de separar los datos provenientes del microcontrolador, para ello el algoritmo que contiene analiza la cadena de entrada, eliminando los caracteres de inicio, fin y separación de trama entre dato y dato, almacenándolos en arreglos temporales de datos de tipo Float, que luego serán pasados al método *IniciarAnimacion()*.

- *IniciarAnimacion;* Este método se encarga de representar la evolución con respecto al tiempo, de los datos adquiridos por los sensores de temperatura y presión, en forma gráfica.

- *PuertoCom_DataReceived;* Este evento se activa o dispara cada vez que el computador detecta que existen datos en uno de sus Puerto Serie, para luego pasar a leerlos y procesarlos respectivamente, el corazón de este evento es importante puesto que aquí se encuentra toda la lógica de adquisición de datos.

El código fuente de esta Clase se adjunta en el apéndice de Anexos, para su mejor entendimiento.

5.4 IMPLEMENTACIÓN COMPONENTE CONTADOR

Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

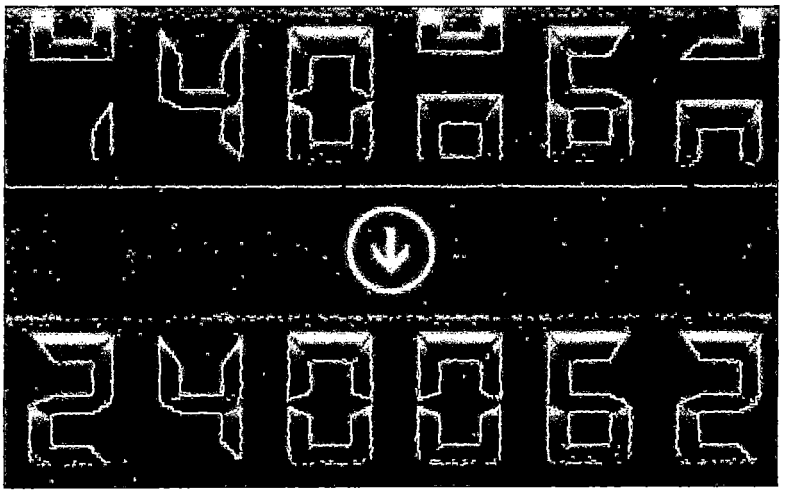
5.4.1 PROPIEDADES

```
public partial class CounterControlMod_0 : UserControl
{
    private int NroInicial = 0;
    private int NroFinal = 0;
    private int AlturaGrid = 0;
    private int GridResaltado;
    private int GridLimiteSuperior = 0;
    private int GridLimiteInferior = 0;
    private double initY;
    private double newY;
    private Canvas DigitCanvas;
    private DispatcherTimer TimAnimacion = new DispatcherTimer();
    private DoubleAnimation dblGridAnim = new DoubleAnimation();
    public  bool Detenido = false;
```

5.4.2 MÉTODOS

```
public  void EstablecerNroAlCargar()...
private void UbicarGridEnPosicionLimite(Grid grd, int LimiteSuperior, int LimiteInferior)...
private void AnimacionMostrarNuevoNumero(Canvas cnv, int LimiteSuperior, int LimiteInferior)...
private void TimeEjecutarAnimacion_Tick(object sender, EventArgs e)...
public  void Iniciar()...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:



Lo más resaltante en la implementación de este control son los métodos *UbicarGridEnPosicionLimite* y *AnimacionMostrarNuevoNumero*, cuyos detalles más sobresalientes pasamos a describir a continuación:

- *UbicarGridEnPosicionLimite;* Este método ubica un numero en la posición que le corresponde, para ello se determina si es menor o mayor que el digito actualmente activo.

- *AnimacionMostrarNuevoNumero;* Este método hace uso del método descrito anteriormente y dependiendo del resultado que este arroja, realiza una pequeña animación, desplazando los demás dígitos hacia arriba o hacia abajo, hasta ubicar el digito requerido en una posición que se pueda visualizar.

5.5 IMPLEMENTACIÓN COMPONENTE DIAL

Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

5.5.1 PROPIEDADES

```
public partial class DialControlMod_0 : UserControl
{
    public double ValorActual...
    public double AnguloInicial...
    public double Circunferencia...
    public double GrosorBorde...
    public double DistanciaTiks...
    public double DistanciaNros...
    public double LongMinTiks...
    public double LongMaxTiks...
    public double ValorMin...
    public double ValorMax...
    public int NroDivisiones...
    public bool ShowNumbers...
    public bool ShowSmallScale...
    public bool ShowLargeScale...
    public double LadoDial...
    public SolidColorBrush ColNumeros...
    public SolidColorBrush ColSmallTiks...
    public SolidColorBrush ColLargeTiks...
    public SolidColorBrush ColBResaltado...
    public SolidColorBrush ColBOpaco...
    public SolidColorBrush ColFResaltado...
    public SolidColorBrush ColFOpaco...
    public SolidColorBrush ColDialResaltado...
    public SolidColorBrush ColDialOpaco...
    public double Angulo...
```

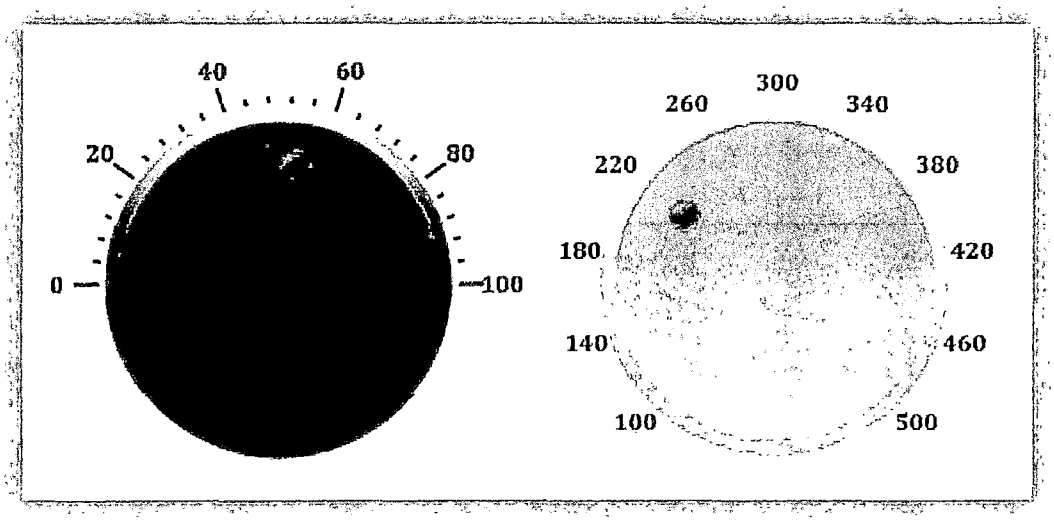
5.5.2 MÉTODOS

```
private void  DibujarDial()...
private Point ObtenerPosicionManija(double Valor)...
private void  DibujarManija()...
```

5.5.3 EVENTOS

```
protected  static void  OnValueChanged(DependencyObject sender, DependencyPropertyChangedEventArgs e)...
private    static void  OnPropertyChange(DependencyObject sender, DependencyPropertyChangedEventArgs e)...
private    void         Dial_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)...
private    void         Dial_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)...
private    void         Dial_MouseMove(object sender, MouseEventArgs e)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:



Lo más resaltante en la implementación de este control es la propiedad *ValorActual* y los métodos *DibujarDial, DibujarManija* y Obtener*PosicionManija*, cuyos detalles más sobresalientes pasamos a describir a continuación:

- *ValorActual;* Esta propiedad contiene el valor actual que muestra el control Dial, luego de mover la perilla alrededor de este.

- *DibujarDial;* Este método dibuja el circulo mayor y en ella se ubican los números, las marcas pequeñas y grandes y también la perilla. Para la ubicación de los números y las marcas alrededor de la circunferencia se hace uso de la trigonometría, en especial del concepto de coordenadas polares; cuya función es determinar puntos alrededor de una circunferencia en base al radio del círculo y un ángulo dado.
- *DibujarManija;* Este método dibuja la perilla que rotara alrededor del circulo mayor. Para ubicar esta perilla en el círculo mayor se hace uso del método Obtener*PosicionManija.*
- Obtener*PosicionManija;* Este método calcula la posición de la perilla dentro del circulo grande, en función a un valor que se le pasa. Para esto también se usa el concepto de coordenadas polares.

5.6 IMPLEMENTACIÓN COMPONENTE DISPLAY

Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

5.6.1 PROPIEDADES

```
public partial class MatrizDePuntosMod_1 : UserControl
{
    public Color ColorHabilitado...
    public bool Habilitado...
    public DireccionMovimiento DireccionDeMovimiento...
    public bool Empezar...
    public bool Detener...
    public Styles Estilo...
    public int NroCaracteres...
    public int NroFilas...
    public int NroColumnas...
    public string DisplayText...
    public double LadoPixel...
    public double RadiusXY...
    public double SeparacionPixeles...
    public enum Styles { Puntos, Rectangulos };
    public enum DireccionMovimiento...;
    Dictionary<char, ulong> CharDict = new Dictionary<char, ulong> { };
    Dictionary<string, ulong> SimbolsDict = new Dictionary<string, ulong> { };
```

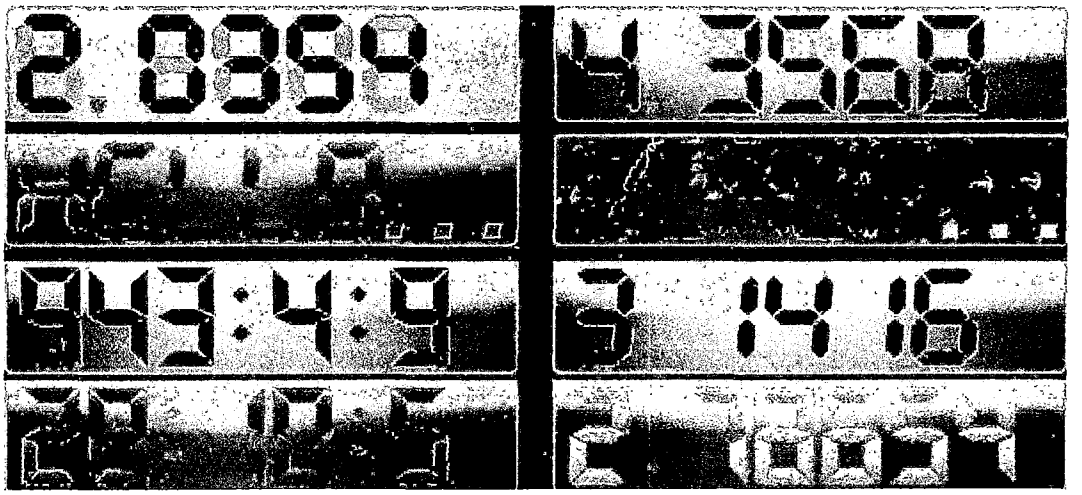
5.6.2 MÉTODOS

```
private void  DiccionarioDeSimbolos()...
private void  DiccionarioDeCaracteres()...
private void  DibujarGridsFondo()...
private void  InicializarGrid(Grid MyGrid)...
public  void  DibujarCaracter(ulong BitsCaracter, double Posicion)...
public  void  DibujarTexto(string Texto)...
private void  Tim_Tick(object sender, EventArgs e)...
public  void  Enabled()...
public  void  StartMove()...
public  void  StopMove()...
```

5.6.3 EVENTOS

```
private static void OnBoolChanged(DependencyObject sender, DependencyPropertyChangedEventArgs e)...
private static void OnPropertyChange(DependencyObject sender, DependencyPropertyChangedEventArgs e)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:



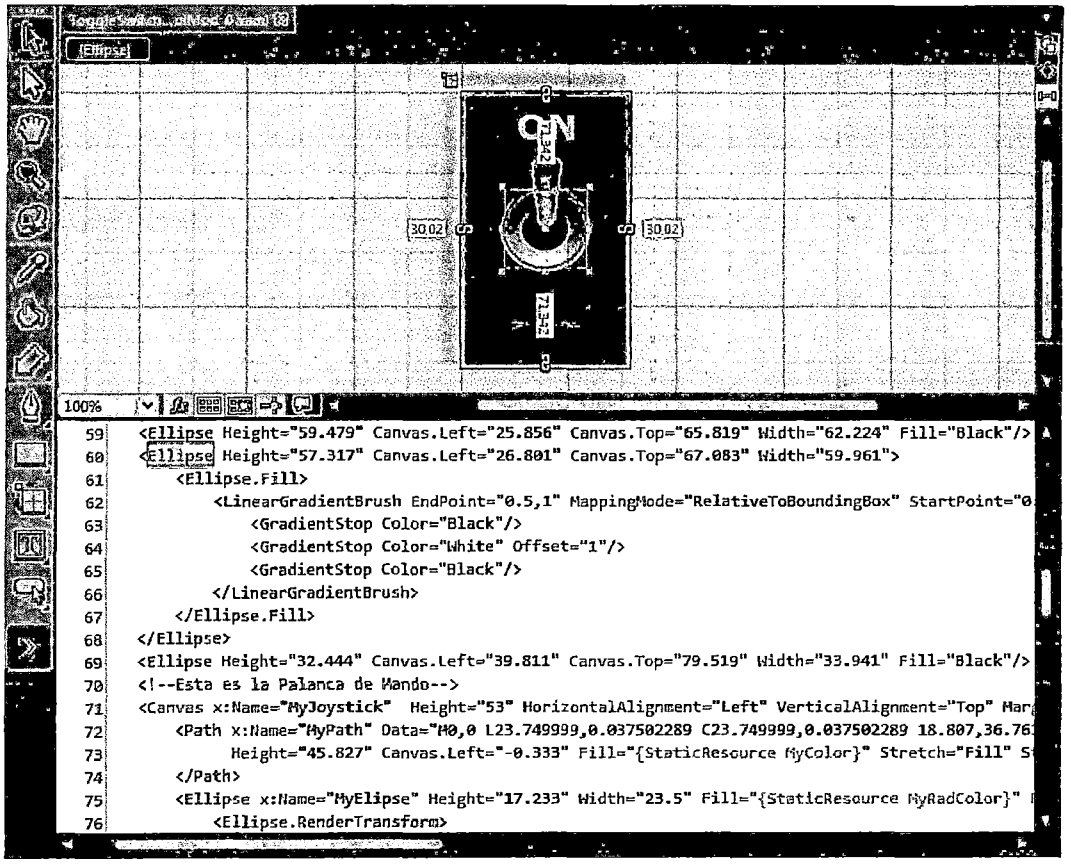
Lo más resaltante en la implementación de este control son los métodos *DiccionarioDeSimbolos, DiccionarioDeCaracteres* y *DibujarCaracter*, cuyos detalles más sobresalientes pasamos a describir a continuación:

- *DiccionarioDeSimbolos y DiccionarioDeCaracteres;* Estos métodos contienen la definición de cómo será la forma de un carácter conocido o especial. Para ello se guarda la configuración de un carácter, en la forma: "A", "87963095043584"; en la cual el número que representa el carácter "A", se transforma a binario y luego el valor de cada bit se representa como un rectángulo o círculo, cuyo color dependerá de si el bit es un 1 o un 0.

- *DibujarCaracter;* Este método dibuja carácter por carácter los dígitos de un mensaje pasado como parámetro. Para lograr esto se hace uso de la configuración que posee el par Clave-Valor, que se almacena en la estructura Dictionary. La forma en la que se dibujan los dígitos es pintando rectángulos de un color u otro, dependiendo de los bits que posee un carácter dado.

5.7 IMPLEMENTACIÓN COMPONENTE INTERRUPTOR

Con la ayuda de la herramienta Expression Blend, el diseño final de este componente luce del siguiente modo:



Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

5.7.1 PROPIEDADES

```
public partial class ToggleSwitchControlMod_0 : UserControl
{
    bool MoverJoystick = false;
    public enum Posicion { Arriba, Abajo };
    Posicion PosActual = Posicion.Arriba;
    public bool IsChecked...
```

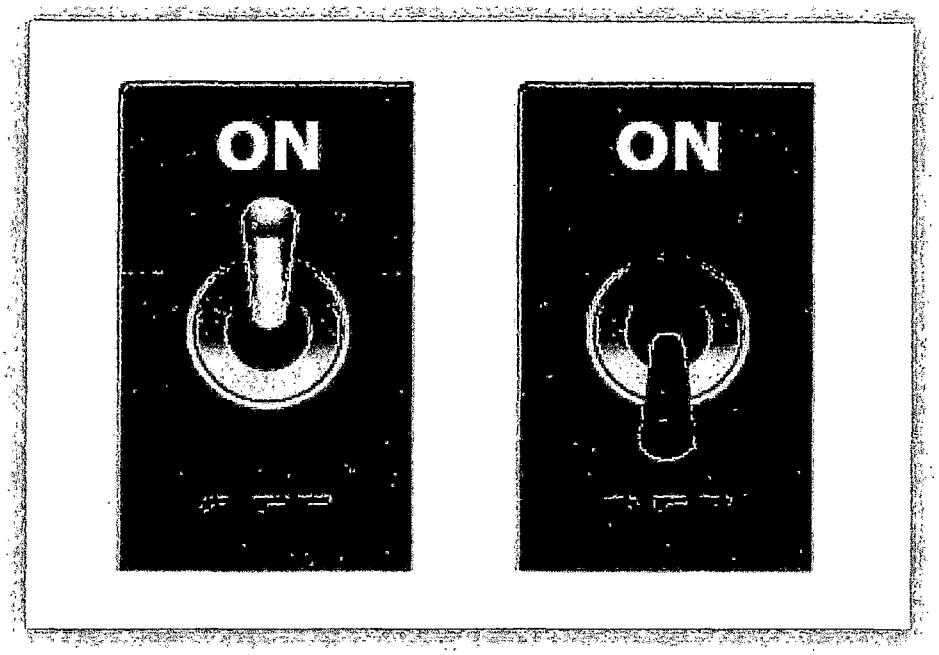
5.7.2 MÉTODOS

```
private void CambiarEstado()
{
    MyJoystick.Cursor = Cursors.Hand;
    MoverJoystick = true;
    if (PosActual == Posicion.Arriba)
    {
        RotateTransform Rotacion = new RotateTransform(180, 0, 0);
        MyJoystick.RenderTransform = Rotacion;
        IsChecked = false;
        PosActual = Posicion.Abajo;
    }
    else
    {
        RotateTransform Rotacion = new RotateTransform(0, 0, 0);
        MyJoystick.RenderTransform = Rotacion;
        IsChecked = true;
        PosActual = Posicion.Arriba;
    }
}
```

5.7.3 EVENTOS

```
protected static void OnCheckedChanged(DependencyObject sender, DependencyPropertyChangedEventArgs e)...
public  void MyJoystick_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)...
public void MyJoystick_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:



Lo más resaltante en la implementación de este control es el método *CambiarEstado*, la propiedad *IsChecked* y el evento *CheckedChanged*, cuyos detalles más sobresalientes pasamos a describir a continuación:

- *IsChecked;* Esta propiedad indica si la manija del control apunta hacia arriba o abajo, y en base a este resultado se realizara una rotación vertical hacia arriba o debajo de la manija del control. Esta variable ésta disponible en todo momento y es importante ya que nos indicara en todo momento en qué estado se encuentra el control, para poder realizar operaciones en base a su valor.
- *CambiarEstado;* Este método se encarga de cambiar la ubicación de la manija del control, hacia arriba o abajo, dependiendo del estado de la variable *IsChecked,* para ello se realiza una rotación vertical .
- *CheckedChanged;* Este evento se activa o dispara cada vez que la manija del control cambia de posición, ya sea apuntando hacia arriba o abajo.

5.8 IMPLEMENTACIÓN COMPONENTE PLOTEADOR DE SEÑALES

Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

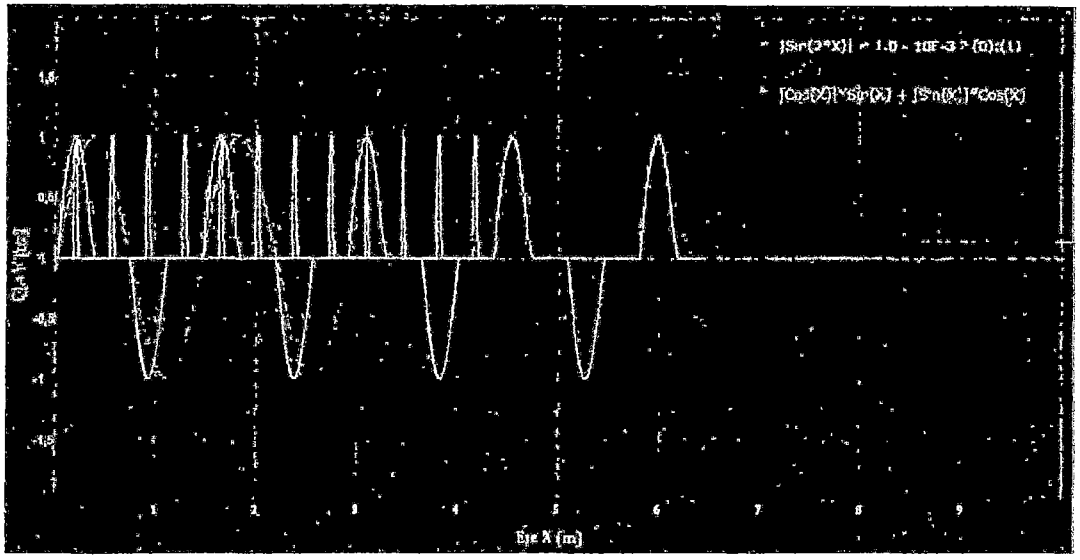
5.8.1 PROPIEDADES

```
public partial class GraphControlMod_1 : UserControl
{
    public enum EstilosDeLinea...;
    public enum OrigenCoordenadas...;
    public string XLabel...
    public string YLabel...
    public Brush YLabelColor...
    public Brush XLabelColor...
    public string GraphLegend...
    public Brush LegendColor...
    public Brush ColorFondo...
    public EstilosDeLinea EstiloLinea...
    public OrigenCoordenadas OrigenDeCoordenadas...
    public double GrosorLineaRect...
    public double SepMarcasX...
    public double SepMarcasY...
    public double EscalaX...
    public double EscalaY...
    public double AvanceX...
    public double AvanceY...
    public double NroDivX...
    public double NroDivY...
    public bool GrillaX...
    public bool GrillaY...
    public Brush ColorRectangulo...
    public Brush ColorLineaCentro...
    public Brush ColorGrilla...
    public Brush ColorNumeroCero...
    public Brush ColorNumeros...
```

5.8.2 MÉTODOS

```
private void XYLabels()...
public void DibujarEjes(Canvas MyCanvas)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:

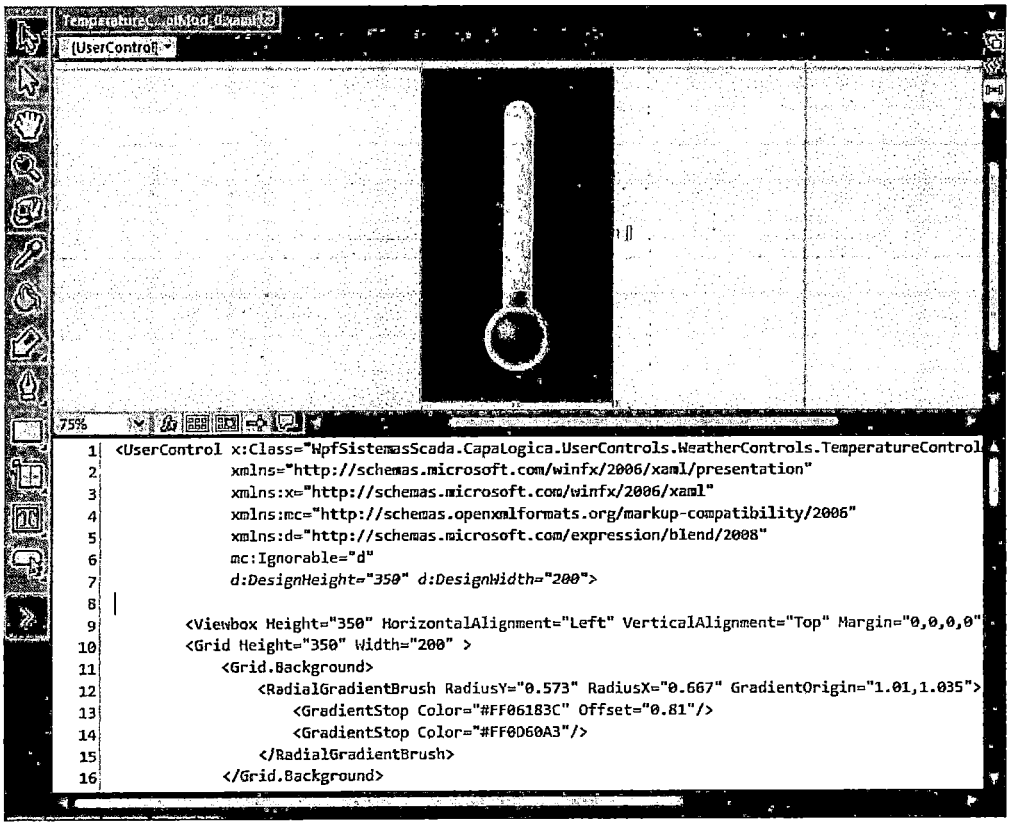


Lo más resaltante en la implementación de este control es el método *DibujarEjes*, y las propiedades *EscalaX* y *EscalaY*, cuyos detalles más sobresalientes pasamos a describir a continuación:

- *EscalaX* y *EscalaY;* Estas propiedades son muy importantes puesto que permiten escalar cualquier tipo de grafico que se pretenda representar en su superficie de dibujo, es decir permite que cada punto de una función cualquiera a graficar, se ubique o coincida con los números del gráfico.
- *DibujarEjes;* Este método se encarga de dibujar las grillas, números, determinar la escala para cada eje de coordenadas, también se encarga de ubicar el origen de coordenadas en un punto determinado del control, dibujar las leyendas, entre otros. Por ejemplo para dibujar las grillas y los números se recorre el control con una estructura repetitiva "for" de izquierda a derecha y de arriba abajo, para ir graficando los puntos y las grillas.

5.9 IMPLEMENTACIÓN COMPONENTE TERMÓMETRO

Con la ayuda de la herramienta Expression Blend, el diseño final de este componente luce del siguiente modo:



Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

5.9.1 PROPIEDADES

```
public partial class TemperatureControlMod_0 : UserControl
{

    public double ValorActual...
    public bool   ReiniciarAgujaInicio...
    public double EscalaBarrido...
    public double CantidadDivisionesMayores...
    public double CantidadDivisionesMenores...
    public double ValorMaximo...
    public double ValorMinimo...
    public int    ValorPrecisionEscala...
    public double EscalaInicialY...
    public Size   TamañoMarcasMayores...
    public Color  ColorMarcasMayores...
    public double EscalaInicialX...
    public double EscalaTextoX...
    public Size   EscalaTamañoTexto...
    public double EscalaTextoFuenteTamaño...
    public Color  EscalaTextoColor...
    public Size   TamañoMarcasMenores...
    public Color  ColorMarcasMenores...
    public double mercurio...
```

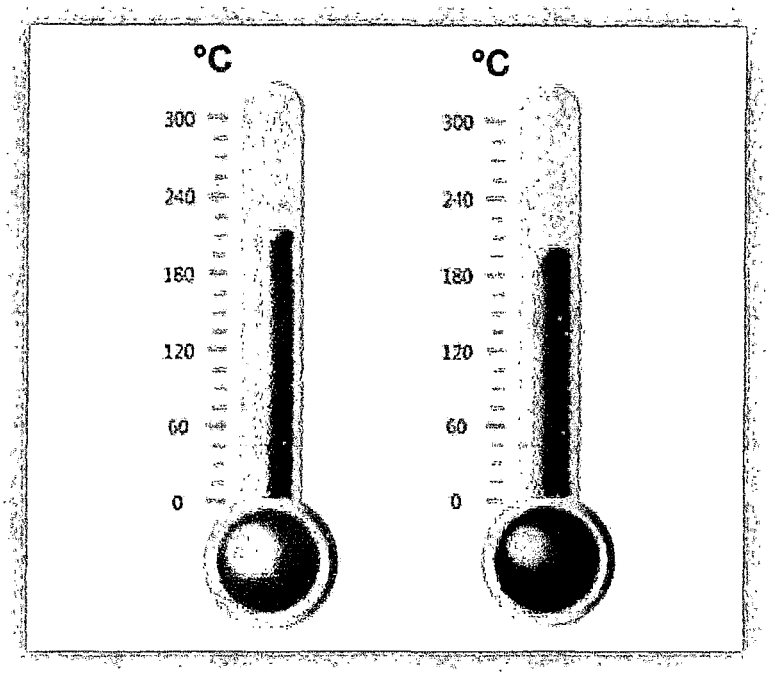
5.9.2 MÉTODOS

```
public  void iniciarectangulo()...
public  void CambiarValorActual(DependencyPropertyChangedEventArgs e)...
public  void AnimarRectangulo()...
private void sb_Completed(object sender, EventArgs e)...
private void DibujarEscala()...
```

5.9.3 EVENTOS

```
private static void OnPropertyChange(DependencyObject sender, DependencyPropertyChangedEventArgs e)...
private       void sb_Completed1(object sender, EventArgs e)...
private static void OnCurrentValuePropertyChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:



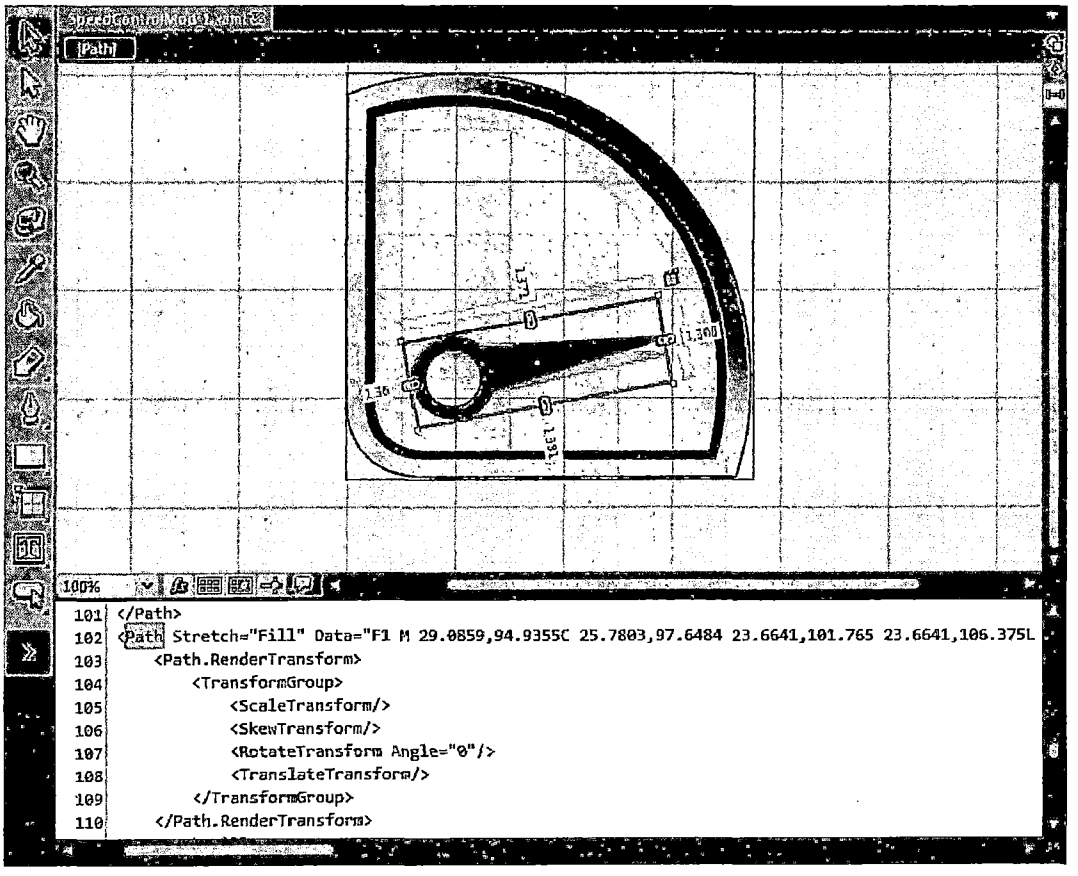
Lo más resaltante de la implementación de este componente se encuentra en los métodos *DibujarEscala*() y *CambiarValorActual(e)*. A continuación una breve explicación del desarrollo de cada uno estos métodos.

- *DibujarEscala();* Este método es implementado en función a las propiedades de dependencia para darle la flexibilidad de cambio. Este método nos permite modelar la escala del termómetro. Se implementó con un modelo anidado de la estructura repetitiva "for", uno para las marcas mayores, y para cada una un "for" para las marcas menores. Las operaciones realizadas en función a las propiedades de dependencia para poder modificar sus propiedades en diseño como en ejecución.

- *CambiarValorActual(e);* Este método consigue capturar el valor actual y el valor anterior para poder realizar la animación en función a estos dos valores, teniendo cuidado de que estos valores no se encuentren fuera de los límites establecidos en la escala (ValorMaximo y ValorMinimo) para lo cual el método hace las validaciones necesarias.

5.10 IMPLEMENTACIÓN COMPONENTE VELOCÍMETRO

Con la ayuda de la herramienta Expression Blend, el diseño final de este componente luce del siguiente modo:



Para implementar la funcionalidad de este componente, a continuación se listaran las propiedades, métodos y eventos que se usaron:

5.10.1 PROPIEDADES

```
public partial class SpeedControlMod_1 : UserControl
{
    public double ValorActual...
    public bool   ReiniciarAgujaInicio...
    public double EscalaBarridoAngulo...
    public double CantidadDivisionesMayores...
    public double CantidadDivisionesMenores...
    public double ValorMaximo...
    public double ValorMinimo...
    public int    ValorPrecisionEscala...
    public double AnguloInicialEscala...
    public Size   TamañoMarcasMayores...
    public Color  ColorMarcasMayores...
    public double RadioEscala...
    public double EscalaRadioTexto...
    public Size   EscalaTamañoTexto...
    public double EscalaTextoFuenteTamaño...
    public Color  EscalaTextoColor...
    public Size   TamañoMarcasMenores...
    public Color  ColorMarcasMenores...
    public double Angulo...
```

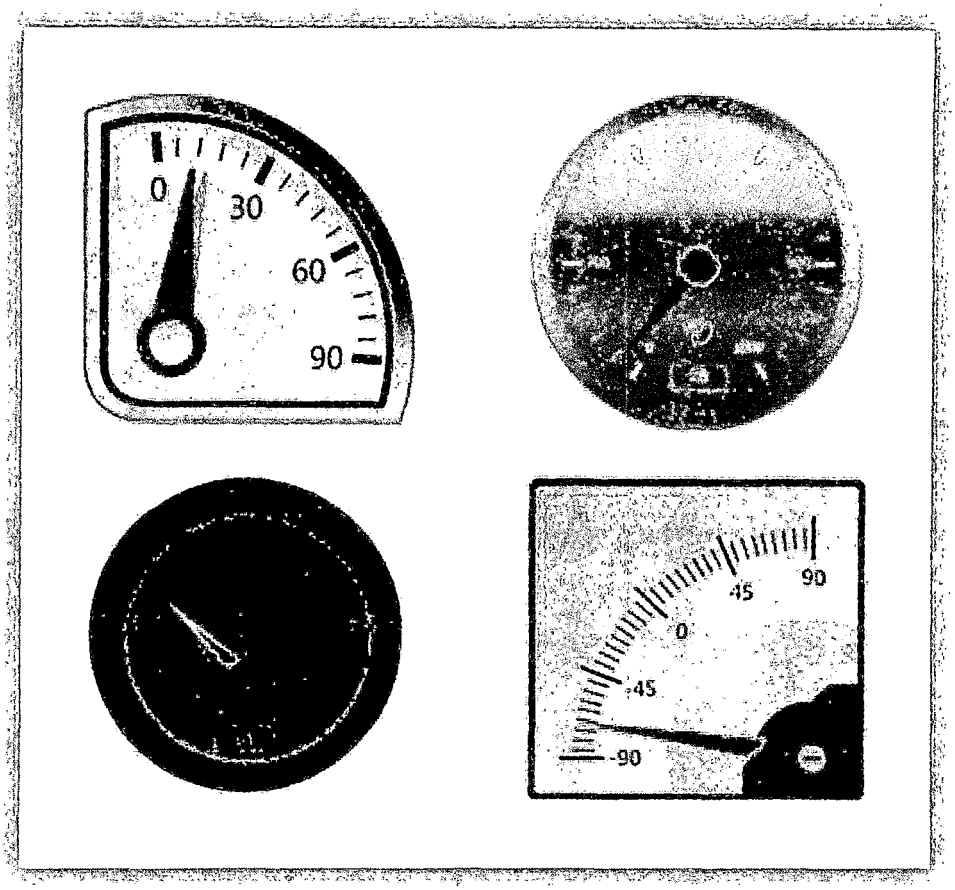
5.10.2 MÉTODOS

```
public  void RotarAngulo()...
public  void CambiarValorActual(DependencyPropertyChangedEventArgs e)...
public  void AnimarAguja(double oldcurrentvalueAngle, double newcurrentvalueAngle)...
public  void DibujarEscalaMoverAguja()...
private void DibujarEscala()...
public  void MoverAguja(double angleValue)...
```

5.10.3 EVENTOS

```
private static void OnPropertyChange(DependencyObject sender, DependencyPropertyChangedEventArgs e)...
private static void OnCurrentValuePropertyChanged(DependencyObject d, DependencyPropertyChangedEventArgs e)...
```

El componente terminado, puesto en funcionamiento se observa en la siguiente imagen:



Lo más resaltante de la implementación de este componente se encuentra en los métodos *DibujarEscala()* y *AnimarAguja(oldcurrentvalueAngle, newcurrentvalueAngle)*. A continuación una breve explicación del desarrollo de cada uno estos métodos.

- *DibujarEscala();* Este método es implementado en función a las propiedades de dependencia para darle la flexibilidad de cambio. Este método nos permite modelar la escala del termómetro. Se implementó usando coordenadas polares para poder graficar cada marca (mayor y menor) de la escala y los numerales al borde del control, en función a las propiedades de dependencia RadioEscala (que representa el tamaño que tendrá el radio de la circunferencia) y AnguloInicialEscala (que convertido a radian representa el valor del Angulo).

- *AnimarAguja(oldcurrentvalueAngle, newcurrentvalueAngle);* Este método lo que hace es realizar la animación de la aguja del control, en función a los parámetros de entrada, que contienen el valor de la posición anterior y el valor de la nueva posición a donde se desplazara la aguja respectivamente. La animación la realiza a la propiedad de rotación de la aguja.

CAPÍTULO

6

PRUEBAS

En el presente capítulo se realizarán distintos tipos de pruebas por cadacontrol desarrollado, alimentándoloscon datos generados por software y por ultimo una prueba en conjunto, con datos adquiridos desde hardware.

La realización de las pruebas en un sistema, no solo es una etapa necesaria dentro del ciclo de vida del desarrollo de software, si no también nos da una idea de la calidad del software desarrollado, pues los resultados obtenidos proporcionan una idea del grado de confianza que brindara el sistema al usuario.

Por tanto las siguientes pruebas a aplicarse nos permitirán determinar si los componentes desarrollados funcionan del modo que se espera o presentan fallas que deben ser corregidas. A continuación se detallan los casos y procedimientos de prueba para los principales componentes del trabajo desarrollado.

Prueba Componente Barras De Progreso		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar valores fuera de rango o valores no válidos.	El Control no debería realizar ninguna acción.	El Control no realizó ninguna acción.
Generar valores válidos, que se encuentren en el rango establecido para el Control.	El Control debería indicar mediante un número el valor proporcionado, además un color ocupa un determinado porcentaje del Control, indicando también de manera gráfica dicho valor.	El valor generado se representa tanto numérica como gráficamente, del modo esperado.

Prueba Componente Clima		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar valores válidos, que se encuentren en el rango establecido para el Control.	El Control debería indicar mediante un determinado grafico si el valor proporcionado corresponde a un clima Soleado, Nublado, Lluvioso, etc.	El valor generado se representa del modo esperado, mediante un tipo de gráfico.

Prueba Componente Contador		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar valores fuera de rango o valores no válidos.	El Control no debería realizar ninguna acción.	El Control no realizó ninguna acción.
Generar valores válidos, que se encuentren en el rango establecido para el Control.	El Control debería indicar mediante un número dicho valor, además un color ocupa un determinado porcentaje del Control, indicando también de manera gráfica dicho valor.	El valor generado se representa tanto numérica como gráficamente, del modo esperado.

Prueba Componente Display		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar valores numéricos o mensajes de texto.	El Control debería representar gráficamente el mismo valor o mensaje.	El parámetro recibido se representa gráficamente del modo esperado.

Prueba Componente Dial		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar valores válidos, que se encuentren en el rango establecido para el Control.	El Control debería posicionar una esfera pequeña, que apunte a dicho valor.	El valor generado se indica posicionando una esfera pequeña, que apunta a dicho parámetro.

Prueba Componente Ploteador De Señales		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar arreglos de datos no validos o ecuaciones matemáticas inconsistentes.	El Control no debería realizar ninguna acción.	El Control no realizó ninguna acción.
Generar arreglos de datos validos o funciones matemáticas válidas.	El Control debería graficar el valor X e Y proporcionados en el arreglo de datos, o generados en base a la función matemática.	La función o el arreglo de datos proporcionado, se grafican del modo esperado.

Prueba Componente Interruptor		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Sincronizar la evolución de un determinado proceso, al estado de este control.	La evolución de un determinado proceso debería estar en marcha si la manija del control indica "Prendido" o debería estar detenido si la manija del control indica "Apagado".	El proceso se detiene si la manija está en el estado "Apagado" y se reanuda o inicia cuando el estado del control es "Prendido".

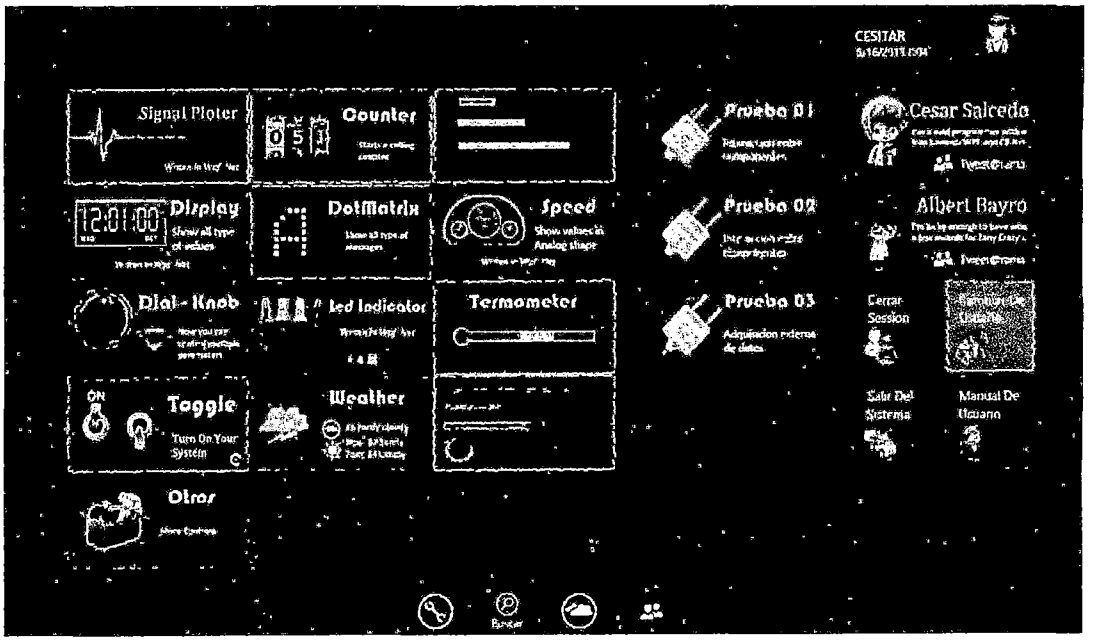
Prueba Componente Batería		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar valores válidos para visualizar el funcionamiento del control.	El control debe de simula el progreso de cargado de una batería haciendo el movimiento hacia la posición que indica del valor generado.	El control simula de forma gráfica el cargado de una batería, indicando el porcentaje de cargado.
Generar valores fuera del rango de los valores permitidos.	El control no debe de realizar ningún funcionamiento.	El control no realiza ninguna simulación después del primer posicionamiento.

Prueba Componente Velocímetro		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar valores óptimos que soporta el control.	El control debe marcar con la aguja el valor generado.	La aguja rota a la posición que indica el valor generado.
Generar valores fuera del rango de los valores permitidos.	El control no debe de marcar el valor que se genera.	El control no realiza ninguna acción después del primer posicionamiento.

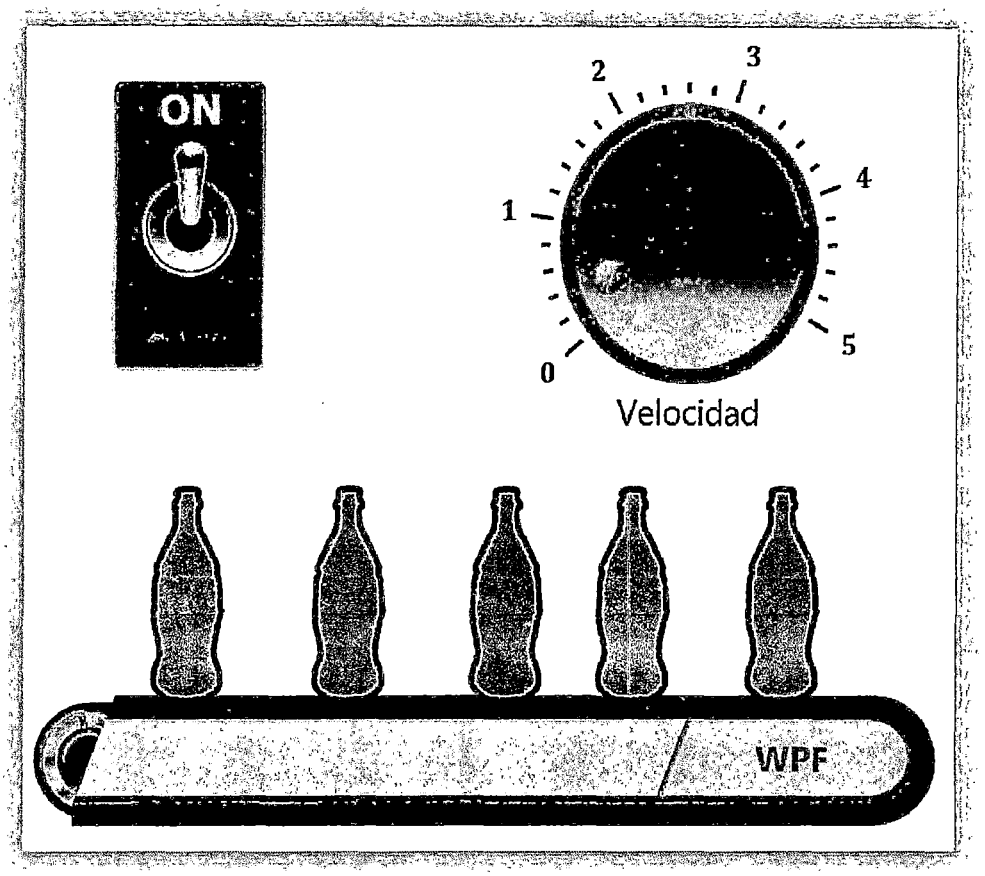
Prueba Componente Termómetro		
Caso De Prueba	**Resultado Esperado**	**Resultado Obtenido**
Generar valores óptimos que soporta el control.	El control debe simular el movimiento del líquido de un termómetro y moverse a la posición que indica el valor generado.	El líquido del control realiza se desplazamiento correcto hacia la posición del valor generado.
Generar valores óptimos por encima y debajo del valor indicador.	El control debe de cambiar el color del líquido del termómetro una vez que el valor generado sea mayor o igual al valor indicador y volver al estado original una vez que este sea menor al valor indicador.	El líquido del control cambia de color una vez que este supera el valor indicador establecido y efectivamente vuelve al estado original una vez que es este es menor al valor indicador.
Generar valores fuera del rango de los valores permitidos.	El control no debe de simular el movimiento del líquido del termómetro.	El control no realiza ninguna acción después del primer posicionamiento.

6.1 APLICACIONES DE EJEMPLO QUE ILUSTRAN EL USO DE LOS COMPONENTES DESARROLLADOS

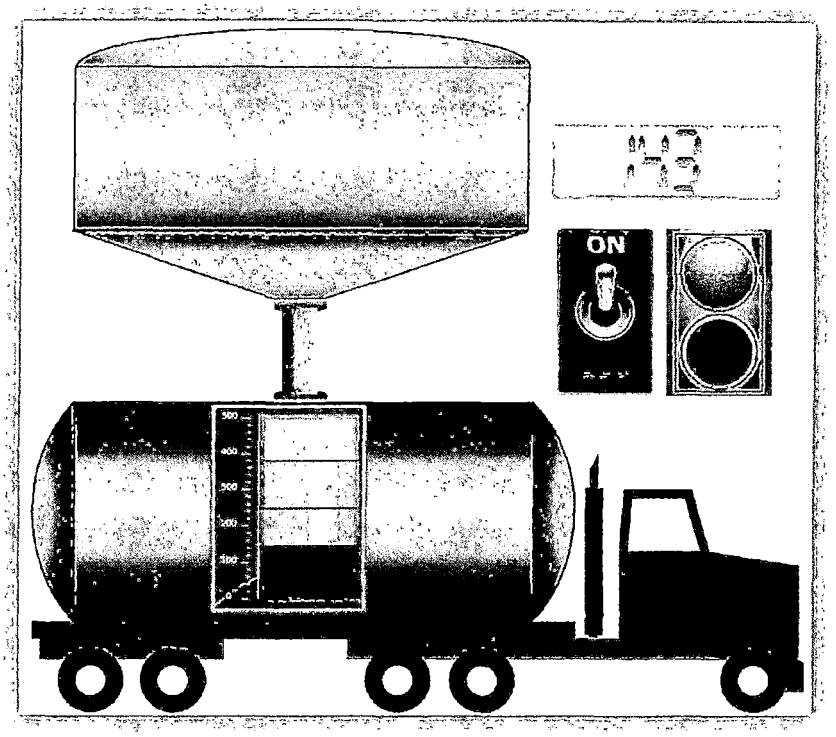
A continuación se muestra el funcionamiento en conjunto de algunos de los controles, que validan los argumentos expuestos en las tablas anteriores:



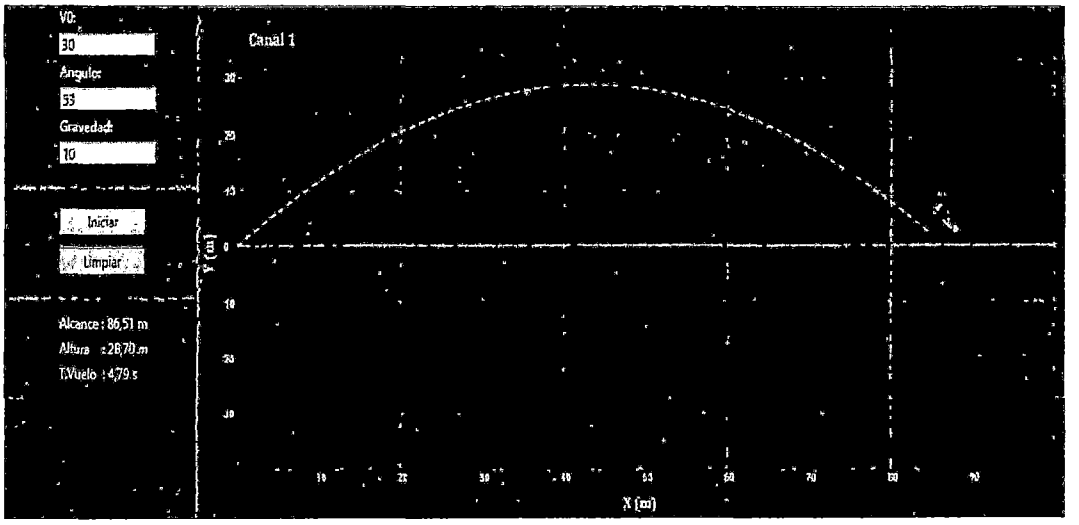
La imagen muestra la ventana principal de la aplicación de ejemplo, el cual contiene un acceso directo del uso de los componentes desarrollados.



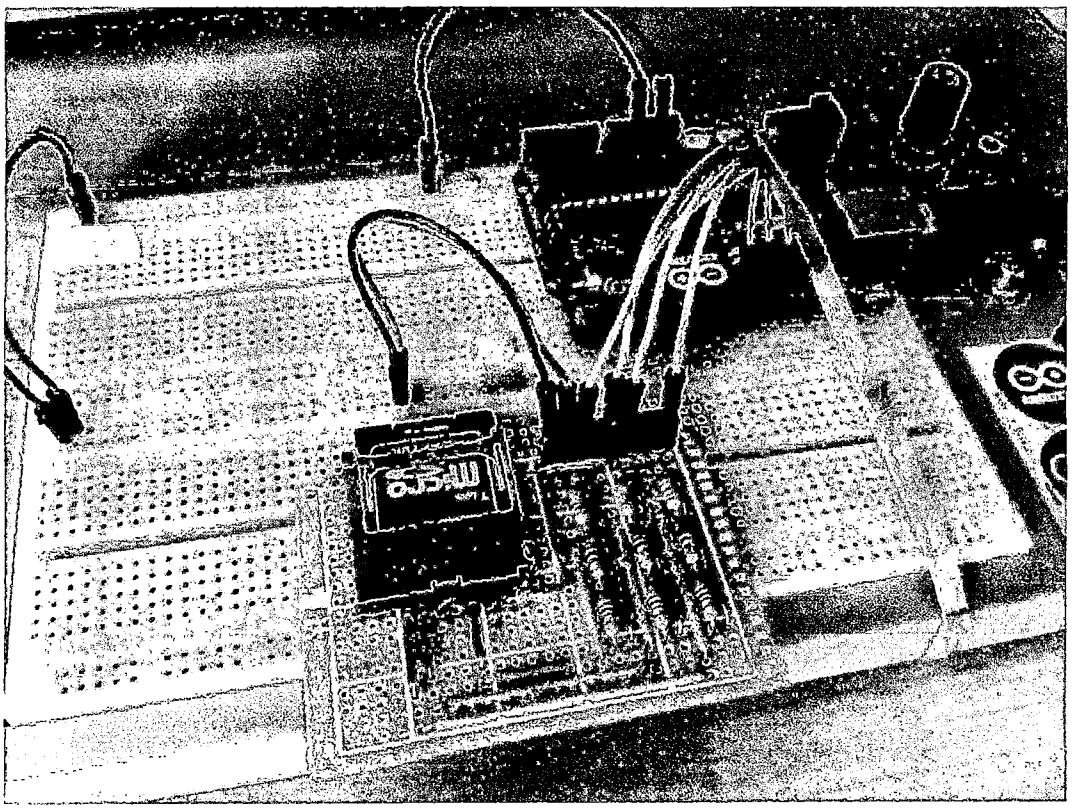
En la imagen podemos observar la simulación de un proceso un tanto común en las instituciones industriales, como es una faja transportadora de botellas, en el cual interactúa con dos de los componentes creados en este proyecto, como es el Componente Interruptor usado para iniciar, pausar y reiniciar el proceso y el Componente Dial que controla la velocidad del proceso en este caso.

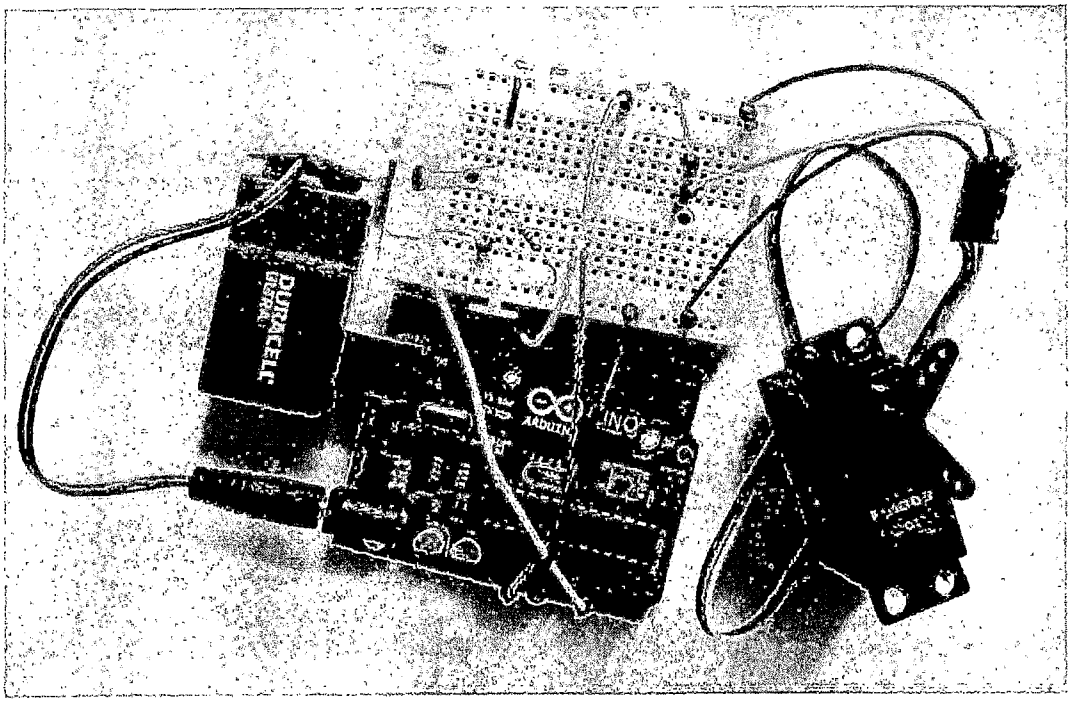


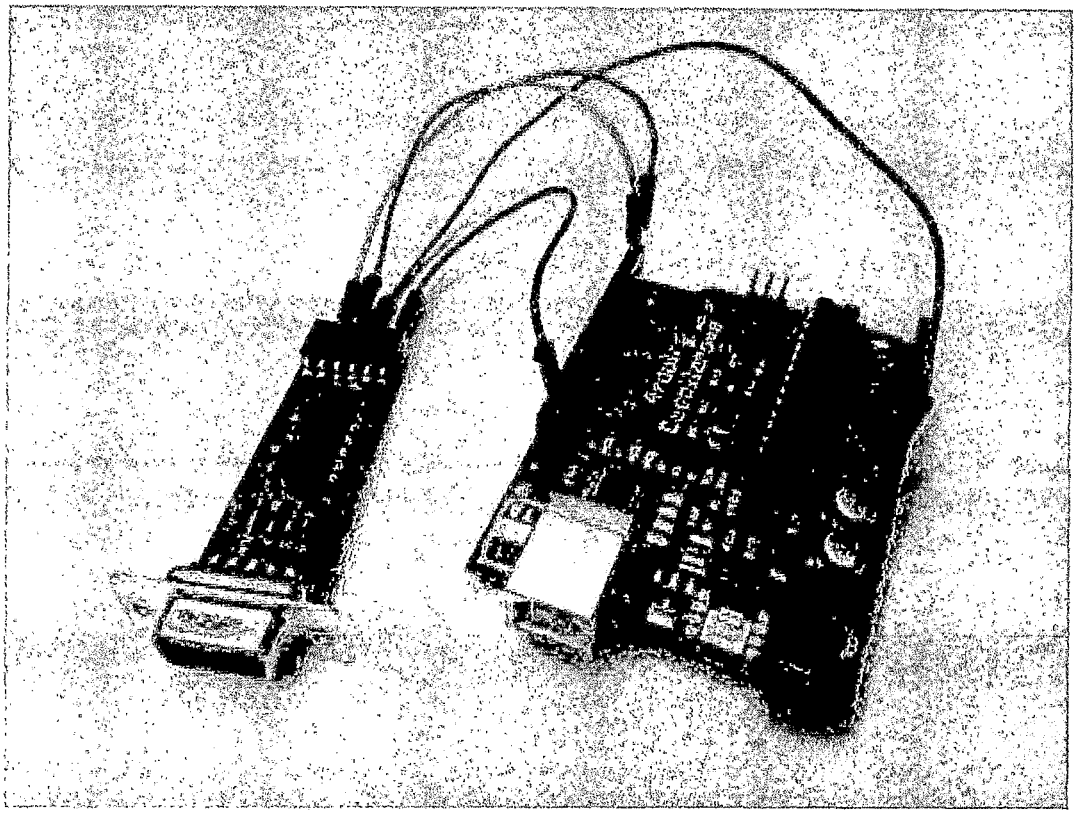
En la imagen podemos observar la simulación de un proceso del llenado de un tanque, en el cual interactúan dos de los componentes creados en este proyecto como es el Componente Interruptor para el iniciar el proceso, detener el proceso si hubiera alguna ocurrencia fuera de lo común y reiniciar el proceso. Otro componente usado es el Componente Display usado para visualizar de manera numérica el valor del nivel del llenado actual del tanque.



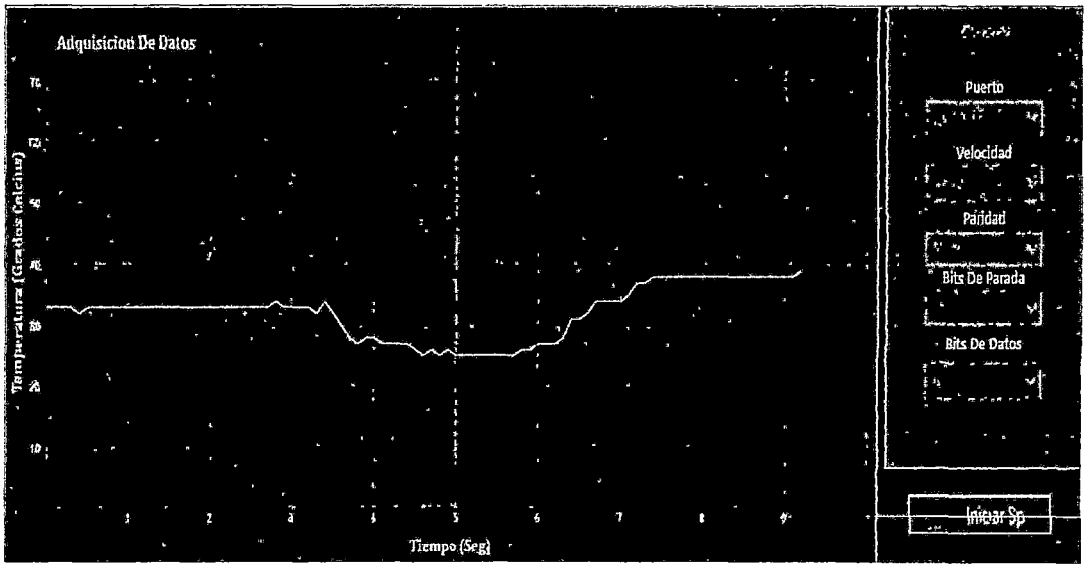
En la gráfica observamos el funcionamiento del Componente Ploteador, que en este caso se usa para la simulación de un fenómeno físico como es el Movimiento Parabólico. Cabe indicar que los valores asignados y los valores calculados son sincronizados con los valores que muestra el control.




DURACELL
ARDUINO



Las imágenes anteriores muestran el circuito que se usó a lo largo del presente proyecto, para generar y adquirir datos de prueba, que envían sus datos al puerto serie de la computadora, a través de un microcontrolador.



La imagen anterior muestra la evolución de la variable temperatura, adquirida desde un sensor de temperatura; con respecto al tiempo.

CONCLUSIONES

Finalizado el presente proyecto los resultados nos permiten llegar a las siguientes conclusiones:

1. El uso de éstos componentes agilizan el tiempo de desarrollo de aplicaciones SCADA, al tener que evitar la construcción de los mismos, lo que implica un ahorro significativo de tiempo.
2. De las pruebas realizadas, se pudo observar el correcto funcionamiento de los componentes desarrollados. Lo cual permitirá que el usuario pueda trabajar de manera rápida y eficiente, con la seguridad de contar con una librería de componentes confiable.
3. Los componentes desarrollados pueden ser usados fácilmente para la implementación de aplicaciones SCADA, en un gran número de industrias.
4. Los componentes desarrollados proporciona información oportuna y precisa de los procesos más frecuentes a las áreas involucradas, ésto ayudará a la oportuna y mejor toma de decisiones por parte de la administración.
5. La utilización de las herramientas como WPF y Expression Blend evitó el hecho de tener que construir algunos componentes desde la parte de código, facilitando su desarrollo de una manera gráfica y permitiendo el enfoque en la lógica del negocio.

RECOMENDACIONES

Gracias a la experiencia adquirida durante la elaboración de este proyecto, se pueden proponer las siguientes recomendaciones:

1. Agregar funcionalidad extra a los componentes ya desarrollados, de modo que les permitan exponer muchas más características.
2. Construir componentes que permitan visualizar datos históricos y estadísticos almacenados.
3. Realizar los ajustes necesarios, para que los componentes puedan funcionar en aplicaciones web y entornos móviles.
4. Realizar los ajustes necesarios, para que los componentes puedan ser migrados a la plataforma Linux, usando la herramienta de desarrollo Mono Develop.
5. Implementar un pequeño aplicativo, para acoplar los controles de forma automática a la barra de herramientas de Visual Studio.

BIBLIOGRAFÍA

TEXTOS

[1] Francisco Charte Ojeda , Anaya Multimedia ,Programación en Microsoft Visual C# .Net

[2] Francisco Javier Ceballos Sierra , Anaya Multimedia ,Enciclopedia de Microsoft Visual C# .Net

[3] Jeff Ferguson, Anaya Multimedia ,La Biblia De C# .Net

[4] Aroa Solana, Luarna Ediciones Noviembre 2010, Desarrollo De Aplicaciones Windows Con WPF 4.0

[5] Francisco Javier Ceballos Sierra, Editorial Ra-Ma 2010 Interfaces Graficas y Aplicaciones Para Internet Con WPF,WCF y Silverlight

[6] Miguel Katrib Mora, Mario Del Valle Matos, Iskander Sierra Zaldivar, Yamil Hernandez Saa, Grupo Weboo ,Bienvenidos Al Curso De Windows Presentation Foundation

[7] Jack Xu,Ph.D, Apress , Practical WPF Charts And Graphics

[8] Jack Xu,Ph.D, UniCAD Publishing , Practical WPF Graphics Programing

[9] Matthew MacDonald, Apress,Pro WPF In C# 2010

[10] Rob Eisenberg,Christopher Bennage, Sams ,Teach Yoursefl WPF In 24 Hours

[11] Chris Andrade,Shawn Livermore,Make Meyers,Scott Van Vliet , Wiley Publishing Inc., Professional WPF Programing In C# 2008

[12] Adam Nathan, Sams Publishing ,Windows Presentation Foundation Unleashed 4.0

[13] Rod Stephens, Wrox ,WPF Programers Reference With C# 2010

[14] Arlen Feldman,Maxx Daymon, Manning Greenwich,WPF In Action With Visual Studio 2008

[15] Sam Noble,Sam Burton, Allen Jones, Apress, WPF Recipies In C# 2008

[16] Andrew Troelsen, Apress ,Pro Expression Blend 4

[17] Brennon Williams, Pearson Education ,Microsoft Expression Blend Unleashed

[18] Gurdy Leete,Mary Leete, Wiley Publishing Inc. , Microsoft Expression Blend Bible

[19] Victor Gaudioso, Apress ,Expression Blend 3 Wirh Silverlight

[20] Aquilino Rodríguez Penin, Editorial Marcombo 2007 , Sistemas

SCADA

[21] Cerro Aguilar Enrique, Ediciones Ceysa 2004, "Comunicaciones Industriales"

[22] Aquilino Rodríguez Penin, Editorial Marcombo Primera Edicion 2008, "Comunicaciones Industriales"

[23] Eduarco Garcia Breijo, Editorial Marcombo Primera Edicion 2008,Compilador C CCS y Simulador Proteus Para Microcontroladores PIC

[24] Juan Ricardo Penagos Plazas, Editorial Micro C,"Cómo programar en lenguaje C los microcontroladores PIC16F88, 16F628A y 16F877A". 2da edición.

DIRECCIONES ELECTRÓNICAS

¿Que son los Microcontroladores? Disponible en:

[1] http://es.wikipedia.org/wiki/Microcontrolador

Controlador Lógico Programable. Disponible en:

[2] http://es.wikipedia.org/wiki/Controlador_l%C3%B3gico_programable

Sistemas de Control Disponible en:

[3] http://www.slideshare.net/posadaco/sistemas-decontrolautomatico

Graphics Device Interface (GDI+). Disponible en:

[4] http://es.wikipedia.org/wiki/Graphics_Device_Interface

Propiedades de Dependencia WPF en:

[5] http://msdn.microsoft.com/es-es/library/ms752914.aspx

[6] http://helpyourselfhere.blogspot.com/p/wpf-fundamentals.html

XAML Disponible en:

[7] http://es.wikipedia.org/wiki/XAML

Storyboard animación WPF en:

[8] http://www.codeproject.com/Articles/364529/Animation-using-Storyboards-in-WPF

DoubleAnimation en WPF Disponible en:

[9] http://msdn.microsoft.com/en-us/library/ms590761.aspx

Aprendiendo Expression Blend Disponible en:

[10] http://expressioniq.com/?p=3512

Controles de Usuario en:

[11] http://msdn.microsoft.com/es-es/library/cc438236%28v=vs.71%29.aspx

Manejo de delegados en:

[12] http://msdn.microsoft.com/es-es/library/ms173171%28v=vs.80%29.aspx

Interfaz Hombre Maquina:

[13] http://www.cibersociedad.net/congres2009/actes/html/com_genesis-virtual-del-3d_818.html

Componentes de Software:

[14] http://arq1software.blogspot.com/2009/12/componentes-de-software.html

ANEXOS

ANEXO Nro. 1

GLOSARIO DE TERMINOS Y ABREVIATURAS

Active X.- Es un entorno para definir componentes de software reusables de forma independiente del lenguaje de programación. Las aplicaciones de software pueden ser diseñadas por uno o más de esos componentes para así proveer su correspondiente funcionalidad.

Actuador.- Es un dispositivo capaz de transformar energía hidráulica, neumática o eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre un proceso automatizado. Este recibe la orden de un regulador o controlador y en función a ella genera la orden para activar un elemento final de control como, por ejemplo, una válvula.

ANSI C.- Se refiere a la familia de normas sucesivas publicadas por el American National Standards Institute(ANSI) para el lenguaje de programación C.

API.- (Application Programming Interface), en español Interfaz de programación de Aplicaciones. Es el conjunto de funciones y procedimientos que ofrece ciertas bibliotecas para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.

Calculo Lambda.- Es un sistema formal diseñado para investigar la definición de función, la noción de aplicación de funciones y la recursión.

CMS. – (Central Monitoring Station), en español Estación Central de Monitoreo.

CLR.- (Common Language Rutime), en español Entorno en tiempo de ejecución de Lenguaje Común.

ECMA.- (European computer Manufacturers Association) Es una organización mundial basadas en membresías de estándares para la comunicación y la información. Fue fundada para estandarizar los sistemas computarizados en Europa. La membresía está abierta a las empresas que producen, comercializan o desarrollan sistemas computacionales o de comunicación en Europa.

DotGNU.- Es parte del proyecto GNU (proyecto con el objetivo de crear un sistema operativo completamente libre) con el fin de proporcionar una alternativa libre para la plataforma de desarrollo Microsoft.net. Otras metas del proyecto es mejorar la compatibilidad con las plataformas diferentes a Windows y a otros procesadores.

EEPROM o E^2PROM. - (Electrically Erasable Programmable Read-Only Memory), en español ROM programable y borrada eléctricamente. Es un tipo de memoria

ROM que puede ser programada, borrada y reprogramada eléctricamente, a diferencia de la EPROM que ha de borrarse mediante un aparato que emite rayos ultravioleta. Son memorias no volátiles.

Factoría.- Ciertos establecimientos instalados en las colonias y dedicados al comercio con la metrópoli. Eran organizaciones de mercaderes que residían en una misma población en territorio colonial, alejado de la metrópoli. Posterior mente se denominó factoría, de forma genérica a cualquier tipo de fábrica o industria.

GDI.- (Graphics Device Interface), en español Interfaz de Dispositivo Grafico.

GPU.- (Graphics Processing Unit), en español Unidad de Procesamiento Grafico.

HTML.- (HyperText Markup Language), en español Lenguaje de Marcado Hipertextual.

IEC.- (International Electrotechnical Commission), en español Comisión Electrónica Internacional. Es una organización de normalización en campos eléctrico, electrónico y tecnologías relacionadas.

ISO.- (International Organization for Standardization), en español Organización Internacional de Normalización. Encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y electrónica.

HMI.- (Human Machine Interface), en español Intefaz Hombre-Maquina.

LabVIEW.- (Laboratory Virtual Instrumentation Engineering Workbench).Es una plataforma y entorno de desarrollo para diseñar sistemas, con un lenguaje de programación visual gráfico.

Librería.- (Biblioteca) Es un colección o conjunto de subprogramas usados para desarrollar software. En lo respecta al lenguaje C++, existen dos tipos de librerías: Estáticas y Dinámicas (DLL).

LINQ.- Language-Integrated Query.

OLE.- (Object Linking and Embedding). Incrustación y enlazado de objetos en español, es el nombre de un sistema de objetos distribuido y un protocolo desarrollado por Microsoft.

OPC.- (OLE for Process Control) Es un estándar de comunicación en el campo del control y supervisión de procesos industriales, basado en una tecnología Microsoft que ofrece una interface común para comunicación que permite que componentes software individuales interaccionen y compartan datos.

PLC.- (Programmable Logic Controller), en español Controlador lógico programable.

Relé.- El relé o relevador es un dispositivo electromecánico. Funciona como un interruptor controlado por un circuito eléctrico en el que, por medio de una bobina y un electroimán, se acciona un juego de uno o varios contactos que permite abrir o cerrar varios circuitos eléctricos independientes.

RTU.- (Remote Terminal Unit), en español Unidad Terminal Remota.

ROM.- (Read-Only Memory), en español Memoria de Solo Lectura. Es un medio de almacenamiento utilizado en ordenadores y dispositivos electrónicos, que permite solo la lectura de la información y no su escritura independiente de la presencia o no de una fuente de energía.

SCADA.- (Supervisory Control And Data Adquisition), en español Supervisión, Control y Adquisición de Datos.

SDK.- (Software Development Kit), en español Kit de Desarrollo de Software.

Third-parties.- (terceros) es el nombre con el cual se conocen a las empresas que desarrollan software libremente para cualquier tipo de plataforma, sin mantener exclusividad con ninguna, aunque también se trata de programas que aceptan archivos de otros programas (third-parties files).

Telemetría.- Es una tecnología que permite la medición remota de magnitudes físicas y el posterior envió de la información hacia el operador del sistema.

UART.- (Universal Asynchronous Receiver-Transmitter), en español Transmisor-Receptor Asíncrono Universal. Este controla los puertos y dispositivos serie. Se encuentra integrada en la placa base o en la tarjeta adaptadora del dispositivo.

WCF.- (Windows Communication Foundation), conocido también como Indigo. Es la plataforma de mensajería que forma parte de la API de la plataforma .NET 3.0. Creada para permitir una programación rápida de sistemas distribuidos y el desarrollo de aplicaciones basadas en arquitectura orientadas a servicios con una API simple; y que puede ejecutarse en una maquina local, una LAN o sobre internet.

WF.- Workflow Foundation.

WPF.- (Windows Presentation Foundation) Es una tecnología de Microsoft, presentado como parte de Windows Vista. Permite el desarrollo de interfaces de interacción en Windows tomando características de aplicaciones Windows y de aplicaciones web.

XAML.- (Extensible Application Markup Language), en español Lenguaje Extensible de Formato para Aplicaciones.

XML.- (eXtensible Markup Language), en español Lenguaje de Marcas Extensible.

XPS.- (XML Paper Specification).

ANEXO Nro. 2

CODIGO CLASE CONEXION

Contiene el código fuente, que realiza la interacción con el Hardware de la aplicación

```
/* ***************************** Atributos ***************************** */

        public enum TipoDeTransmision { Text, Hex }
        private TipoDeTransmision aTipoTransmision = TipoDeTransmision.Text;
        private TextBox aDisplayWindow;
        private string aTasaDeBaudios;
        private string aParidad;
        private string aBitsDeParada;
        private string aBitsDeDatos;
        private string aNombrePuerto;
        private SerialPort aPuertoCom;
        private Canvas aSuperficieDibujo;

        public CConeccionManager(Canvas pMyCanvasDibujo, TranslateTransform
        pMiTraslacion, double pEscalaX, double pEscalaY)
        {
            aSuperficieDibujo = pMyCanvasDibujo;
            aMiTraslacion = pMiTraslacion;
            aEscalaX = pEscalaX;
            aEscalaY = pEscalaY;
            // Iniciamos Parametros del Puerto Serie
            aTasaDeBaudios = string.Empty;
            aParidad = string.Empty;
            aBitsDeParada = string.Empty;
            aBitsDeDatos = string.Empty;
            aNombrePuerto = string.Empty;
            aDisplayWindow = null;

            aPuertoCom = new SerialPort()
            {
                Handshake = Handshake.None,
                ReadTimeout = 500,
                WriteTimeout = 500
            };
              aPuertoCom.DataReceived                    +=                    new
              SerialDataReceivedEventHandler(PuertoCom_DataReceived);
        }

/*  ****************************  Propiedades  ****************************
*/

        public Canvas SuperficieDibujo
        {
            get { return aSuperficieDibujo; }
            set { aSuperficieDibujo = value; }
        }
```

```
        public string TasaDeBaudios
        {
            get { return aTasaDeBaudios; }
            set { aTasaDeBaudios = value; }
        }
        public string Paridad
        {
            get { return aParidad; }
            set { aParidad = value; }
        }
        public string BitsDeParada
        {
            get { return aBitsDeParada; }
            set { aBitsDeParada = value; }
        }
        public string BitsDeDatos
        {
            get { return aBitsDeDatos; }
            set { aBitsDeDatos = value; }
        }
        public string NombrePuerto
        {
            get { return aNombrePuerto; }
            set { aNombrePuerto = value; }
        }
        public TipoDeTransmision TipoTransmision
        {
            get { return aTipoTransmision; }
            set { aTipoTransmision = value; }
        }
        public TextBox DisplayWindow
        {
            get { return aDisplayWindow; }
            set { aDisplayWindow = value; }
        }

    /* ***************************** Metodos ****************************** */

        public void CargarBitsDeDatos(object obj)
        {
            ((ComboBox)obj).Items.Add("7");
            ((ComboBox)obj).Items.Add("8");
            ((ComboBox)obj).Items.Add("9");
        }
        public void CargarBitsDeParidad(object obj)
        {
            foreach (string str in Enum.GetNames(typeof(Parity)))
            {
                ((ComboBox)obj).Items.Add(str);
            }
        }
        public void CargarBitsDeParada(object obj)
        {
            foreach (string str in Enum.GetNames(typeof(StopBits)))
            {
                ((ComboBox)obj).Items.Add(str);
            }
        }
        public void CargarNombrePuertos(object obj)
        {
            foreach (string str in SerialPort.GetPortNames())
            {
```

```
                ((ComboBox)obj).Items.Add(str);
            }
        }
        public void CargarTasaDeBaudios(object obj)
        {
            ((ComboBox)obj).Items.Add("2400");
            ((ComboBox)obj).Items.Add("4800");
            ((ComboBox)obj).Items.Add("9600");
            ((ComboBox)obj).Items.Add("19200");
            ((ComboBox)obj).Items.Add("38400");
        }
        private byte[] HexadecimalABytes(string msg)
        {
            msg = msg.Replace(" ", "");
            byte[] comBuffer = new byte[msg.Length / 2];
            for (int i = 0; i < msg.Length; i += 2)
            {
                comBuffer[i / 2] = (byte)Convert.ToByte(msg.Substring(i, 2),
16);
            }
            return comBuffer;
        }

        private string BytesAHexadecimal(byte[] comByte)
        {
            StringBuilder builder = new StringBuilder(comByte.Length * 3);
            foreach (byte data in comByte)
            {
                builder.Append(Convert.ToString(data,          16).PadLeft(2,
'0').PadRight(3, ' '));
            }
            return builder.ToString().ToUpper();
        }

        public void DisplayData(string DatosPuertoSerie)
        {
            char Item;
            int i = 0;
            bool Detener = false;
            #region MyRegion

            while (Detener == false)
            {
                //Armamos el Primer Valor
                Item = DatosPuertoSerie[i];
                if (Item == '*')
                {
                    string Valor1 = string.Empty;
                    i++;
                    while (char.IsNumber(DatosPuertoSerie[i]))
                    {
                        Valor1 += DatosPuertoSerie[i].ToString();
                        i++;
                    }
                    //Al final vaciamos esto a una lista
                    Datos1.Add(Convert.ToDouble(Valor1));
                }

                // Empezamos a armar el siguiente valor
                Item = DatosPuertoSerie[i];
                if (Item == '@')
                {
```

```
                    string Valor2 = string.Empty;
                    i++;
                    while (char.IsNumber(DatosPuertoSerie[i]))
                    {
                        Valor2 += DatosPuertoSerie[i].ToString();
                        i++;
                    }
                    Datos2.Add(Convert.ToDouble(Valor2));
                }
                //Procesamos el caracter de finalizacion
                if (Item == '+')
                {
                    i++;
                    if (i >= DatosPuertoSerie.Length)
                    {
                        Detener = true;
                    }
                }
            }
            #endregion

            IniciarAnimacion();
        }
        public void AbrirPuerto()
        {
            try
            {
                //Comprobamos si el puerto ya está abierto,si es asi lo
cerramos
                if (aPuertoCom.IsOpen == true)
                { aPuertoCom.Close(); }
                //Establecemos las propiedades de nuestro objeto SerialPort
                aPuertoCom.BaudRate = int.Parse(TasaDeBaudios);    //BaudRate
                aPuertoCom.DataBits = int.Parse(BitsDeDatos);    //DataBits
                aPuertoCom.StopBits  =  (StopBits)Enum.Parse(typeof(StopBits),
BitsDeParada);    //StopBits
                aPuertoCom.Parity     =     (Parity)Enum.Parse(typeof(Parity),
Paridad);
                aPuertoCom.PortName = NombrePuerto;   //PortName

                aPuertoCom.Open(); //Ahora abrimos el puerto
            }
            catch (Exception ex)
            {
                DisplayData(ex.Message);
            }
        }
        public void EnviarDatos(string msg)
        {
            switch (TipoTransmision)
            {
                case TipoDeTransmision.Text:
                    {
                        try
                        {
                            if      (!(aPuertoCom.IsOpen      ==      true))
aPuertoCom.Open();
                            aPuertoCom.Write(msg);//Enviamos  el  mensaje  al
puerto
                            DisplayData(msg + "\n");//Mostrar el mensaje
                        }
                        catch (Exception)
```

```
                    {
                        MessageBox.Show("Error abriendo/escribiendo el
puerto serie :: ", "Error!");
                    }

                } break;
            case TipoDeTransmision.Hex:
                {
                    try
                    {
                        //Convertimos el mensaje a un arreglo de bytes
                        byte[] MsgAEnviar = HexadecimalABytes(msg);
                        //Enviamos el mensaje al puerto
                        aPuertoCom.Write(MsgAEnviar, 0,
MsgAEnviar.Length);
                        //Volvemos a convertir a hexadecimal y luego lo
mostramos
                        DisplayData(BytesAHexadecimal(MsgAEnviar) + "\n");
                    }
                    catch (FormatException ex)
                    {
                        //Visualización del mensaje de error
                        DisplayData(ex.Message);
                    }
                    finally
                    {
                        aDisplayWindow.SelectAll();
                    }
                } break;
        }
    }

    /* ***************************** Eventos ****************************
*/

    //Este evento se lanzara siempre que haya datos esperando en el buffer
    //Representa el método que controlará el evento de recepción de datos
de un objeto SerialPort
    private void PuertoCom_DataReceived(object sender,
SerialDataReceivedEventArgs e)
    {
        //Determinar el modo de que el usuario selecciono (binario /
cadena)
        switch (TipoTransmision)
        {
            case TipoDeTransmision.Text:
                {
                    //Leemos los datos que estan esperando en el búfer
                    string msg = aPuertoCom.ReadExisting();
                    // Mostrar los datos al usuario
                    if (msg != String.Empty)
                    {
                        DisplayData(msg);
                    }
                } break;
            case TipoDeTransmision.Hex:
                {
                    //Recuperamos el número de bytes de datos en el búfer
                    int NroBytesALeer = aPuertoCom.BytesToRead;
```

```
                        //Creamos una matriz de bytes para guardar los datos
en espera
                        byte[] comBuffer = new byte[NroBytesALeer];
                        //Leemos los datos y los almacenamos
                        aPuertoCom.Read(comBuffer, 0, NroBytesALeer);
                        //mostrar los datos al usuario
                        DisplayData(BytesAHexadecimal(comBuffer) + "\n");
                    } break;
                }
            }
```

ANEXO Nro. 3

CODIGO DEL MICROCONTROLADOR

Contiene el código que está en el microcontrolador, el cual envía dos datos, separados por un delimitador (@), hacia la aplicación mediante el puerto serie.

```
#include <16F877a.h>
#FUSES HS,NOWDT
#use delay(clock=20M)
#use rs232(baud=9600,parity=N, xmit=pin_c6, rcv=pin_c7,bits=8)
#include <lcd.c>
int16 espera=0;
int8 valor[4]; //vector en el cual almacenaremos los datos
recibidos
char valor1[1];
int8 i=0;
int8 v1=0;
float dato1;
int8 v2=0;
float dato2;
int8 inicio=0x2d;
int8 final=0x23;

#int_rda
 RDA_isr()
   {
    valor[0]=getc();
if (valor[0]==0x0b)
{
 for(i=1;i<3;i++) //almacena 2 datos del mod usart
    {
     valor[i]=getc();
    }
   }
 }

void main() {
    set_tris_b(0b00000000);
    set_tris_a(0b00000011);
    output_a(0x00);
    output_B(0x00);

    setup_adc(adc_clock_internal);//oscilador interno para la
conversion
    setup_adc_ports(ALL_ANALOG );//
    enable_interrupts(INT_RDA);
    enable_interrupts(global);
    lcd_init();
while(true)
{
```

```
            set_adc_channel(0);//activación del canal analógico
            delay_ms(10);
            v1=read_adc();//lectura del valor digital
            dato1=v1*1.0;
            lcd_gotoxy(1,1);
            printf(lcd_putc,"ADC0=%4.0f",dato1);

            set_adc_channel(1);//activación del canal analógico
            delay_ms(10);
            v2=read_adc();//lectura del valor digital
            dato2=v2*1.0;
            lcd_gotoxy(1,2);
            printf(lcd_putc,"ADC1=%4.0f",dato2);

            if (espera==10){ //valor único para cada soccer
               putc('*');
               delay_us(10);
               printf("%U",v1);
               delay_us(10);
               putc('@');
               printf("%U",v2);
               delay_us(10);
               putc('+');
               espera=0;
             }
             espera++;
          }
}
```