Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Modernización del control de las redes de seguimiento de aguas superficiales SAICA y REACAR

Autor: Valeriano Sanabria Ballesteros

Tutor: María Teresa Ariza Gómez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Modernización del control de las redes de seguimiento de aguas superficiales SAICA y REACAR

Autor:
Valeriano Sanabria Ballesteros

Tutor:
María Teresa Ariza Gómez
Profesor titular

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Carrera: Modernización del control de las redes de seguimiento de aguas superficiales SAICA y REACAR

Autor: Valeriano Sanabria Ballesteros

Tutor: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A mi mujer e hijo

A mis padres

Agradecimientos

Ahora o nunca.

Quiero dar las gracias a mis compañeros de universidad, **mis amigos**, porque hemos compartido juntos todos estos años, ayudándonos, apoyándonos y animándonos cuando más lo necesitamos.

Gracias **a mi madre**, que no ha dejado pasar un día sin preguntarme cuándo voy a terminar el proyecto. Crea que va a ser la persona más feliz cuando lo haga.

También quiero agradecer a la empresa donde trabajo, **Tekromin S.L.**, la cual considero un segundo hogar, por permitirme presentar uno de los proyectos que he realizado con ellos sin poner ninguna objeción. **A mis compañeros**, por sus ayudas y enseñanzas. **A mi jefe**, por su comprensión.

A **Confederación Hidrográfica del Guadiana**, el cliente para el que se ha realizado este proyecto. A los trabajadores con los que he colaborado en éste y otros proyectos.

Dar las gracias a **María Teresa Ariza Gómez** por ayudarme a cerrar de una vez esta etapa de mi vida. Desde el primer encuentro ha sido comprensiva y colaboradora.

Muchas gracias **a mi familia**, por innumerables motivos.

Y, especialmente, gracias **a mi pareja**, mi complemento, mi vida, porque me lo ha dado todo. Por cuidar de nuestro hijo para que yo tuviese tiempo para concentrarme y escribir.

Muchas gracias a todas las personas que me han ayudado tanto estos años.

Valeriano Sanabria Ballesteros.

Resumen

El Sistema Automático de Información de la Calidad de las Aguas (SAICA) y la Red de Estaciones Automatizadas de Control de Aguas Residuales (REACAR) son dos de las redes de seguimiento de la cuenca del Guadiana. Estas redes permiten monitorizar el estado del agua en diferentes puntos estratégicos a lo largo de la cuenca. De esta manera, se obtiene una herramienta fundamental para la gestión y conservación de los recursos acuáticos.

La labor realizada en este proyecto ha sido la modernización del control de las redes SAICA y REACAR de la Confederación Hidrográfica del Guadiana. Los sistemas precedentes se encontraban con partes obsoletas y otras eran mejorables, por lo que una actualización de estos sistemas era vital para continuar con una buena gestión.

La solución desarrollada consiste en la toma de datos referente a la calidad del agua en las estaciones de medida de las redes, el almacenamiento y validación de estos datos, y, finalmente, la publicación de los datos válidos para su posterior estudio.

Abstract

The Automatic Water Quality Information System (SAICA) and the Network of Automatic Residual Water Quality Control Stations (REACAR) are two monitoring networks of the Guadiana basin. These networks monitor the water conditions at several strategic points along the basin. This offers a fundamental tool for the management and the conservation of aquatic resources.

The work done in this project has been the modernization of SAICA and REACAR networks of the Confederación Hidrográfica del Guadiana. The previous systems became obsolete and improvable, therefore the system upgrade was necessary to continue with a good management.

The developed solution consists in the data acquisition from the measurement stations, the data storage and validation, and, finally, the valid data publication for further studies.

Índice

Índice de Tablas

Índice de Figuras

1 INTRODUCCIÓN

Esta memoria describe la solución desarrollada para la modernización de las redes SAICA y REACAR de la cuenca hidrográfica del Guadiana. El proyecto ha sido realizado por el alumno dentro de la empresa Tekromin S.L. para su cliente, Confederación Hidrográfica del Guadiana[1]. La solución consiste en la adquisición de datos de las estaciones de las redes, el almacenamiento, la validación y la publicación de estos datos.

En este capítulo se exponen los objetivos, antecedentes y motivaciones para su realización; se describe la arquitectura de la solución y se resume el contenido de la memoria.

1.1 Objetivos

El fin de este proyecto es proveer a la Confederación Hidrográfica del Guadiana de un sistema automático de almacenamiento de datos de las redes SAICA y REACAR, y de herramientas para el tratamiento y la publicación de estos datos.

- La adquisición de datos por parte del sistema debe ser totalmente automática. El usuario sólo ha de introducir en el programa las direcciones IP de las estaciones de medidas. Las direcciones IP serán almacenadas y podrán ser modificadas por el usuario cuando sea necesario.
- Una vez realizada la lectura de datos de una estación, los datos serán almacenados en una base de datos para su posterior tratamiento.
- La aplicación mostrará los datos almacenados al usuario en una gráfica de tendencia.
- El usuario podrá validar los datos directamente en la gráfica.
- La interfaz gráfica debe ser fácil de usar y visualmente agradable para que no canse al técnico que trabaje en la validación de datos y realización de informes.
- El usuario podrá elegir qué variables quiere validar, seleccionar el periodo de tiempo a valorar y dar un concepto de validación a los valores pertenecientes a la selección realizada.

[1] El alumno cuenta con la autorización de la empresa y del cliente para la presentación del proyecto como Proyecto Final de Carrera.

- La gráfica de tendencia debe ser navegable. Debe tener la opción de representar desde 1 hora a 1 mes completo y avanzar y retroceder en el tiempo de manera sencilla. También se podrá ir a una fecha concreta de manera directa.
- El usuario podrá seleccionar qué estaciones de medidas están operativas. Las estaciones de medida se irán modificando una vez se haya desarrollado la solución e irán estando a disposición del nuevo sistema gradualmente.
- Desde Internet se tendrá acceso a datos válidos almacenados en la base de datos a través de una publicación Web para su consulta.
- La publicación Web sólo ofrecerá los datos válidos de la semana anterior a la semana en que se realiza la consulta.
- Los datos de la publicación Web mostrará los datos en una gráfica de líneas de tendencia.
- El técnico del centro de control podrá elegir qué datos se publican.
- Se debe desarrollar una herramienta para que el técnico pueda extraer los datos que necesite de la base de datos en un formato que pueda trabajar para realizar informes sin que exista la posibilidad de alterar los datos almacenados.

Durante el desarrollo del proyecto, el cliente puede querer modificar algunos elementos de la solución. Estos cambios se estudiarán y se llevarán a cabo si son posibles y se consideran una mejora.

1.2 Antecedentes

El principal objetivo de las Redes de Seguimiento de Aguas Superficiales es generar la información necesaria para llevar a cabo una gestión eficaz del estado de las masas de aguas. La información generada es utilizada por los gestores responsables de la toma de decisiones ya que permite evaluar la efectividad de las medidas adoptadas y el grado de cumplimiento de los objetivos marcados. Estas redes permiten dar respuestas a ciertas necesidades:

- Conocer el estado de las masas de aguas.
- Generar información básica para adoptar estrategias orientadas a combatir la contaminación.
- Vigilar de manera sistemática la calidad de las aguas afectadas por vertidos urbanos o industriales.
- Controlar el efecto que produce la emisión de sustancias en el medio acuático.
- Controlar que las masas de aguas destinadas a determinados usos cumplen con los requisitos de calidad necesarios.
- Evaluar la efectividad de las medidas adoptadas para el control y reducción de la contaminación.

Dos de estas redes de seguimiento de aguas superficiales son el Sistema Automático de Información de Calidad de las Aguas (red SAICA) y la Red de Estaciones Automatizadas de Control de Agua Residuales (red REACAR).

La red SAICA tiene el principal objetivo de producir información continua y trasmitirla al Ministerio de Medio Ambiente y a los centros de proceso de datos de las Confederaciones Hidrográficas. La red SAICA proporciona ayuda e información sobre la situación de la calidad de las aguas continentales superficiales con la siguiente finalidad:

- Proporciona información cualitativa de la contaminación detectada y su evolución en el tiempo, analizando las curvas de tendencia.
- Complementa las redes de control periódico de la calidad de las aguas existentes.
- Disuade de vertidos intencionados.
- Monitoriza en tiempo real, permitiendo actuaciones inmediatas de alerta a las captaciones existentes.

- Facilita el control y seguimiento a corto plazo del vertido.

Por otro lado, la red REACAR está diseñada para ofrecer información sobre los vertidos urbanos o industriales autorizados. El objetivo es controlar que estos vertidos cumplen determinadas normas y se ajustan a los parámetros necesarios de calidad de agua. Si el vertido no cumple estas premisas necesarias, la autorización es retirada.

1.2.1 Red SAICA

En la Confederación Hidrográfica del Guadiana, la red SAICA dispone de 29 estaciones de medidas repartidas estratégicamente a lo largo de toda la cuenca. La distribución de estas estaciones es la tabla **(Tabla 1-1)** y la figura **(Figura 1-1)** siguiente:

Tabla 1-1. Estaciones de medida de la red SAICA

Número	Estación	Río	Municipio
401	Entrerríos	Zújar	Villanueva de la Serena
402	Medellín	Guadiana	Medellín
403	Valverde de Mérida	Guadiana	Valverde de Mérida
404	Bonhabal	Bonhabal	Almendralejo
406	Guadajira	Guadajira	Talavera la Real
407	Valdelacalzada	Guadiana	Valdelacalzada
408	Badajoz	Guadiana	Badajoz
409	Río Caliente	Río Caliente	Cortegana
410	Canal del Piedras	Canal de riego del Piedras	Cartaya
411	Ruidera	Guadiana	Ruidera
412	Villarrubia de los Ojos	Gigüela	Villarrubia de los Ojos
413	Luciana	Guadiana	Luciana
414	Puebla de Don Rodrigo	Guadiana	Puebla de Don Rodrigo
415	Guadalmez	Guadalmez	Guadalmez
416	Jabalón	Jabalón	Moral de Calatrava
417	Bañuelos	Bañuelos	Fernancaballero
418	Amarguillo	Amarguillo	Herencia
419	Alarcos	Guadiana	Ciudad Real
420	Almadén	Valdeazogues	Almadén
421	El Viso	Guadarramilla	El Viso
422	Ruecas	Ruecas	Rena
423	Búrdalo	Búrdalo	Santa Amalia
424	Torremayor	Guadiana	Torremayor
425	Puebla de la Calzada	Guadiana	Valdelacalzada
426	Talavera la Real	Guadiana	Talavera la Real
427	Pesquera	Guadiana	Badajoz
428	Benavides	Guadiana	Badajoz
429	Olivenza	Guadiana	Olivenza
430	Bocachanza	Guadiana-Chanza	El Granado

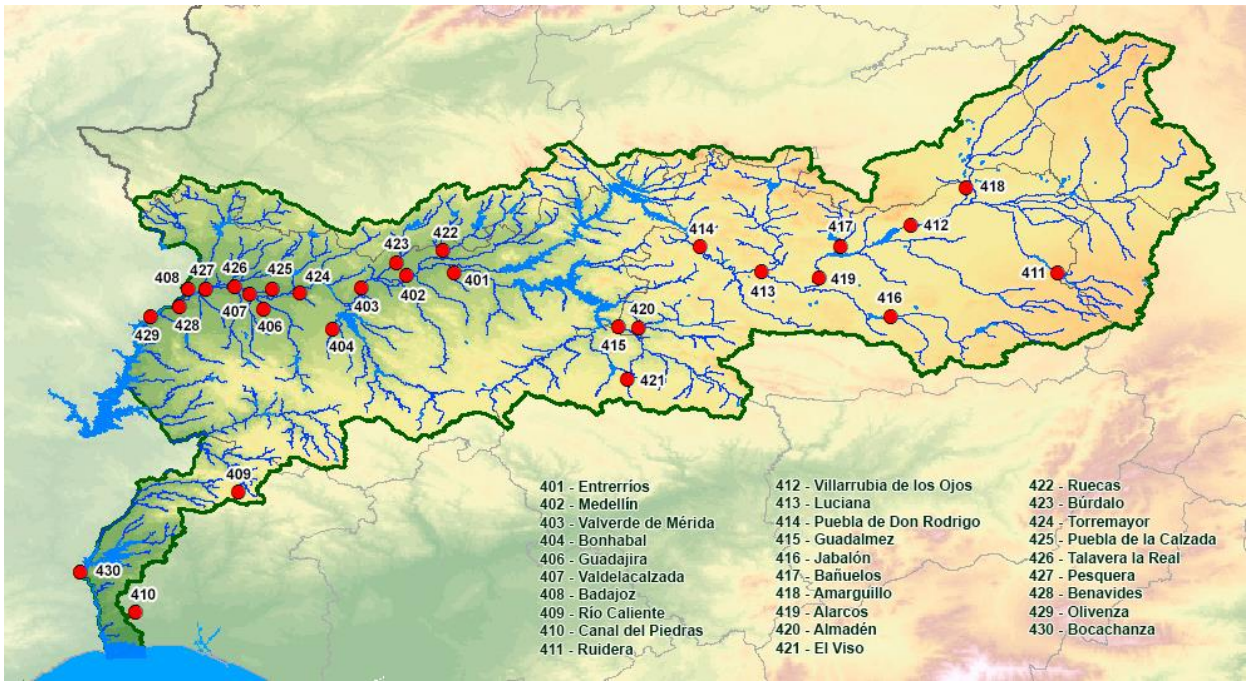

Figura 1-1. Estaciones de medidas de la red SAICA

Las estaciones de medidas del sistema de control previo de la red SAICA obtenían los datos de los sensores existentes y los almacenaban en un PC a la espera de ser consultados por el centro de control. Cada estación estaba compuesta por:

- PC industrial.
- Router 3G.
- Radio/Módem UHF TETRA MDT-400 de Teltronic.
- Red de sensores.

El PC de una estación de medidas se conectaba a la red interna de Confederación Hidrográfica del Guadiana de dos formas, mediante un túnel VPN a través del router 3G y con una conexión radio a la red TETRA (Terrestrial Trunked Radio) perteneciente a la confederación. De esta forma, el PC quedaba dentro de la intranet con dos direcciones IP distintas, una para cada medio de conexión. El esquema de la red de una estación SAICA se muestra en la figura **(Figura 1-2)**.

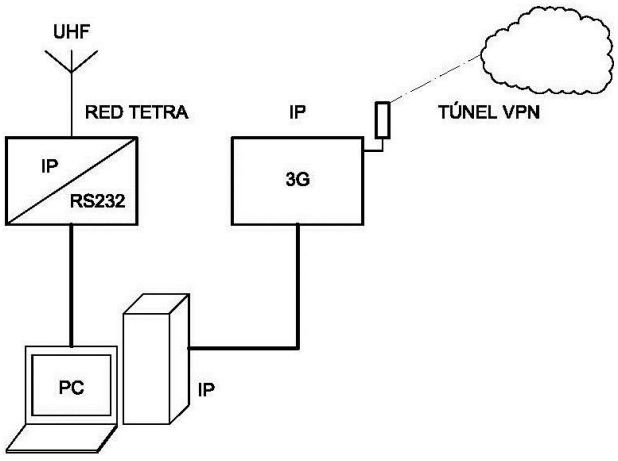

Figura 1-2. Red de una estación de medida SAICA

El centro de control de la red SAICA obtiene los datos de sus estaciones de medida y los incluía en una base de datos. Estaba formado por dos PC industriales. El primero realizaba la conexión con las estaciones de medidas, extraía los datos y los almacenaba en la base de datos que se encontraba en el segundo equipo. En este PC, además de la base de datos, existía un programa de validación llamado Waternet que mostraba los datos gráficamente. El técnico que trabaja en el centro de control podía consultar los datos mediante Access de Microsoft Office y generar informes con ellos.

1.2.2 Red REACAR

La red REACAR estaba dividida en dos partes: REACAR Oriental y REACAR Occidental. Las estaciones de medida están localizadas **(Figura 1-3)** en las siguientes poblaciones **(Tabla 1-2)**:

Tabla 1-2. Estaciones de medidas de la red REACAR

REACAR Oriental	REACAR Occidental
Alcázar de San Juan	Badajoz
Daimiel	Miajadas
Ossa de Montiel	Don Benito
Pozoblanco	Villafranca
Valdepeñas	Jerez de los Caballeros
Villarrobledo	Montijo
Villarrubia de los Ojos	Olivenza
Villarta de San Juan	Almendralejo

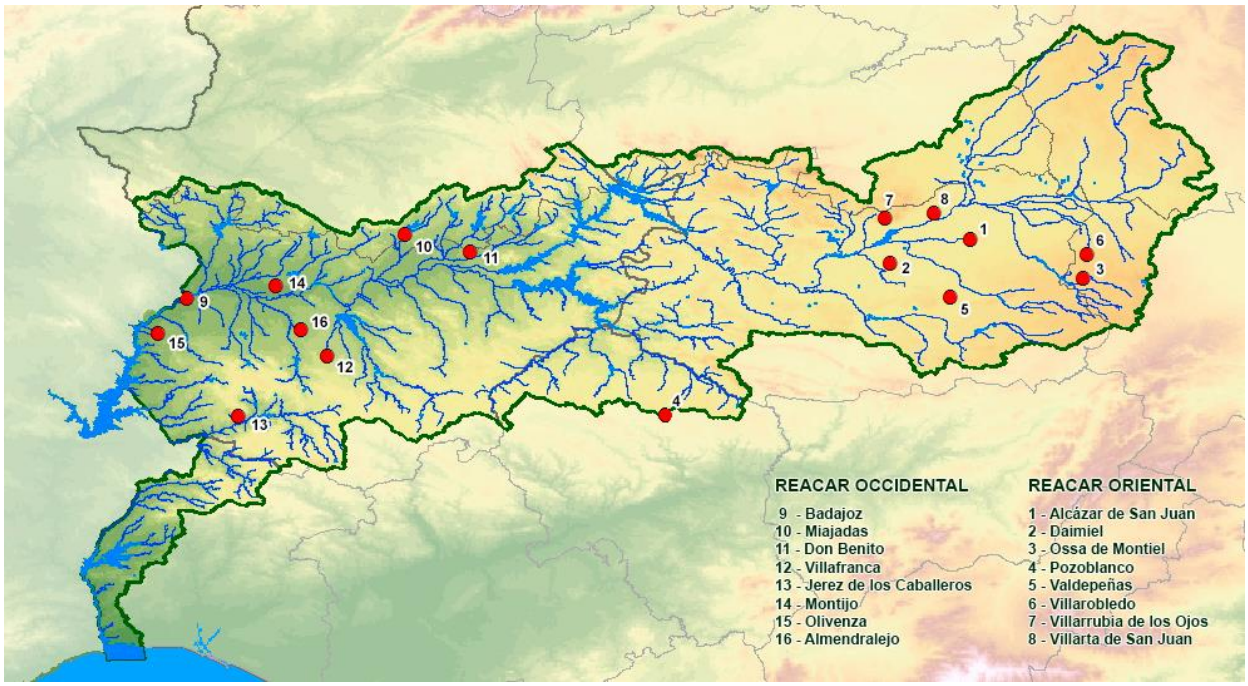

Figura 1-3. Estaciones de medidas de la red REACAR

Las estaciones de medida de la red REACAR estaban compuestas por los siguientes equipos:

- Equipo de medidas AQUATEST de ADASA.
- Módulo GSM Siemmens MC35i.
- Cámara IP.
- Router 3G.

Los datos necesarios eran obtenidos por el centro de control directamente del equipo de medidas. Este equipo era accesible gracias al módulo GSM con el que estaba conectado mediante un cable serie RS-232. La cámara IP y el router formaban parte del sistema anti intrusismo de la red REACAR. La red de una estación REACAR viene representada en la siguiente figura **(Figura 1-4)**:

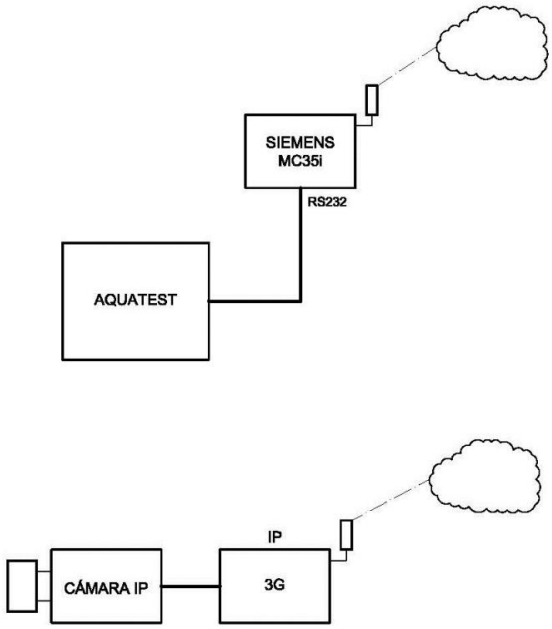

Figura 1-4. Red de una estación de medidas REACAR

El centro de control de la red REACAR disponía de dos PC industriales conectados cada uno a un módulo GSM y a un router 3G. Cada equipo se encargaba de una parte de la red REACAR (Oriental y Occidental). En cada PC estaba instalado un SCADA que realizaba la conexión con los dispositivos AQUATEST en las estaciones vía GSM, extraían los datos, los mostraba en pantalla y generaba informes de cada estación de medida Las conexiones del centro de control se muestran a continuación **(Figura 1-5)**:

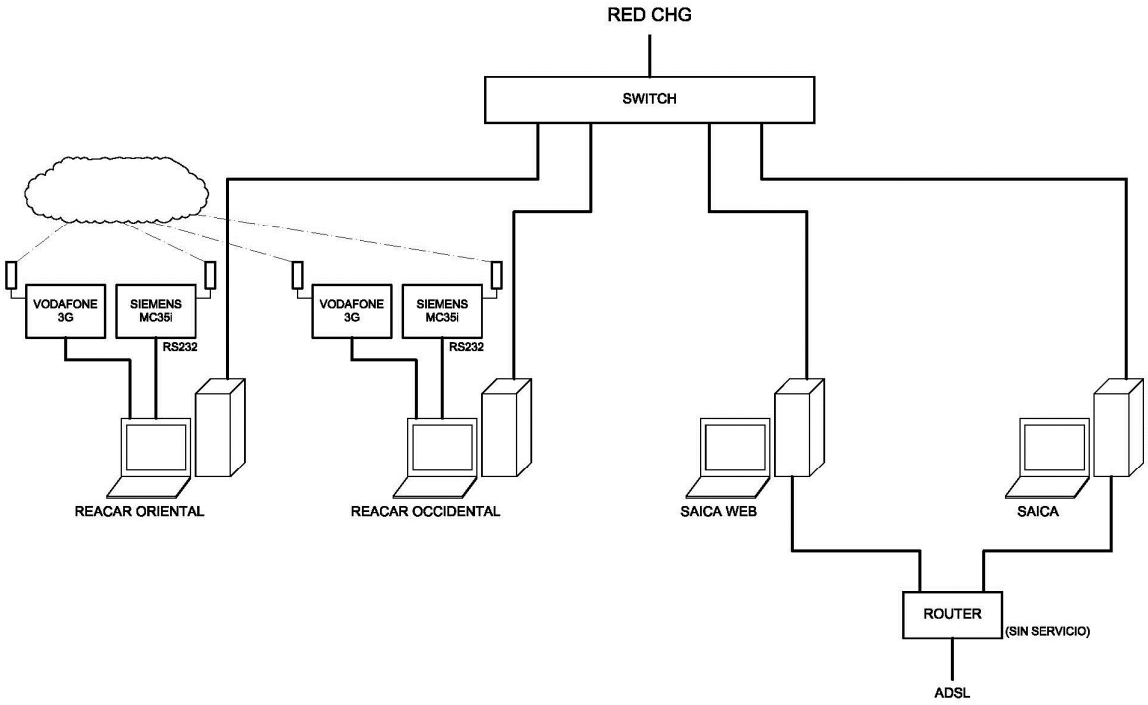

Figura 1-5. Esquema de la red del centro de control

1.3 Motivación

La motivación de realizar este proyecto viene por dos caminos.

En primer lugar, existe la necesidad de modernizar los sistemas de control de las redes SAICA y REACAR puesto que los sistemas precedentes estaban dando cada vez más problemas y se estaban quedando obsoletos por las nuevas necesidades. Los problemas que presentaban los antiguos sistemas eran:

- La comunicación por los módulos GSM no era suficientemente estable.
- Se requiere tomar medidas de otros parámetros y el anterior sistema no permitía añadirlos.
- El manejo de la interfaz era rudimentario.
- El sistema no era escalable.
- La configuración del centro de control de la red SAICA no era estable.
- Los SCADAs de la red REACAR dejaban de funcionar a los pocos segundos de iniciar su ejecución, por lo que seguir trabajando con ellos era inviable.
- El servicio de publicación Web de los datos recabados no estaba funcionando en ninguna de las redes.

Debido a todo lo anterior, se sugirió al cliente desarrollar un sistema que integrase todo lo aprovechable de los sistemas anteriores (la red de sensores de las estaciones, las comunicaciones TETRA y los túneles VPN) y solucionase todos los inconvenientes mostrados, sustituyendo así los sistemas de control de las redes SAICA y REACAR.

Por otro lado, en el ámbito de desarrollo profesional y personal, este proyecto me permite aplicar conocimientos adquirido durante mi etapa universitaria, aprender nuevos lenguajes de programación, iniciarme en el trabajo con bases de datos, familiarizarme con un entorno de trabajo que será muy útil en el futuro más próximo y aprender a relacionarme con los clientes, algo fundamental para el éxito profesional.

1.4 Arquitectura de la Solución.

La solución se compone de dos partes claramente separadas. Por un lado se encuentra la configuración de las estaciones de medidas de las redes SAICA y REACAR, y en el otro lado se halla la arquitectura del centro de datos tal y como se representa en la figura **(Figura 1-6)**.

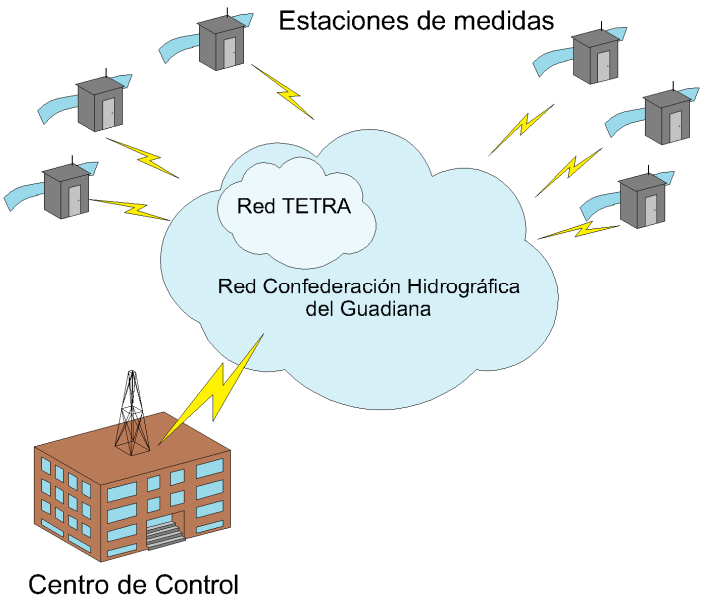

Figura 1-6. Arquitectura de las redes SAICA y REACAR

Las estaciones, a su vez, están subdivididas en la obtención de los datos de la red de sensores, la preparación de los datos para la comunicación y la propia comunicación con el centro de control como puede verse en la

figura **(figura 1-7)**. La ejecución de este proyecto no incluye la obtención de datos de la red de sensores, ni la preparación de los datos para la comunicación puesto que esta parte ha sido desarrollada paralelamente por otro compañero.

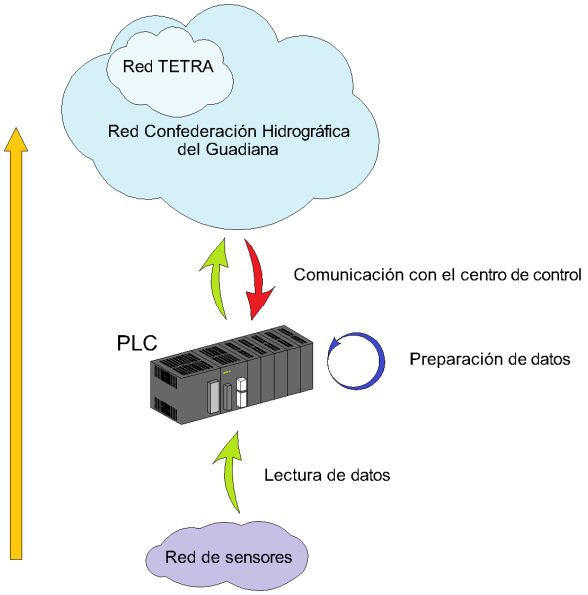

Figura 1-7. Arquitectura de una estación de medidas.

Al ser la red SAICA y la red REACAR dos redes de seguimiento diferentes, es necesario separarlas en el centro de control. Por ello, cada red dispondrá de su propio servidor. Aun así, la arquitectura del servidor de la red SAICA y la red REACAR es la misma. En la siguiente figura **(Figura 1-8)** se muestra la configuración de un servidor del centro de control.

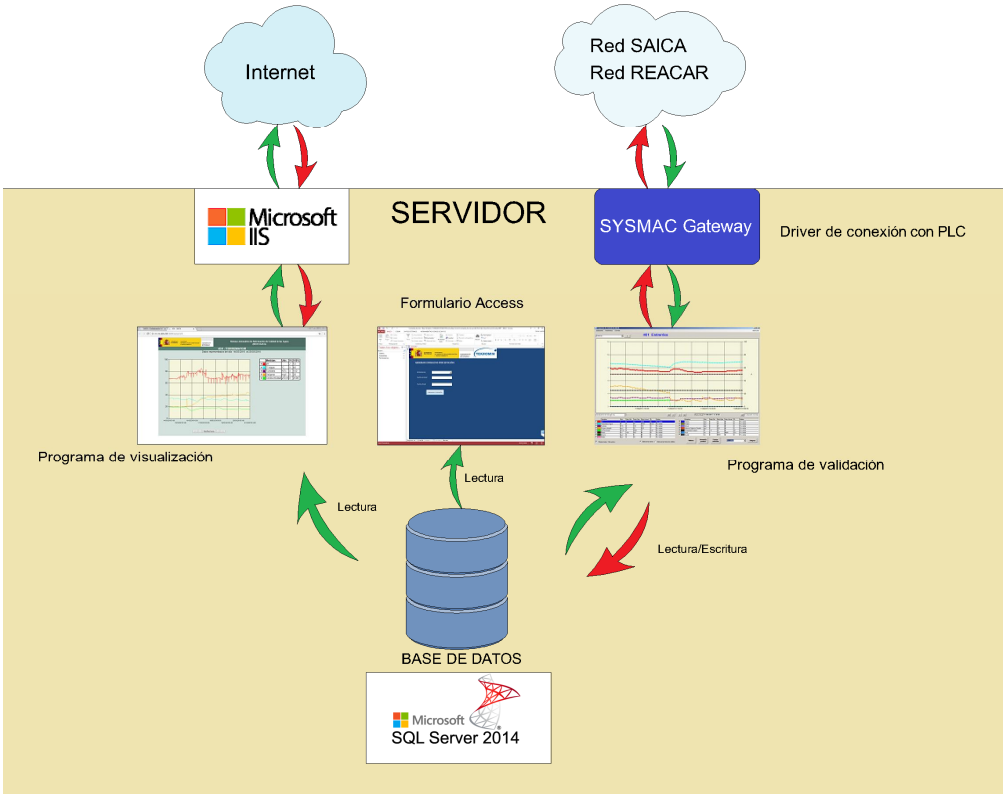

Figura 1-8. Arquitectura de un servidor del centro de control

Dentro del servidor se identifican las siguientes partes de la solución:

- **Base de datos:** Es la encargada de almacenar los datos de forma segura para que éstos no se pierdan ni sean alterados.
- **Programa de validación:** Este programa tiene varios cometidos. En primer lugar, realiza las conexiones con las estaciones de medidas de la red, toma los datos de ellas y los introduce correctamente en la base de datos. Paralelamente, muestra los datos contenidos en la base de datos al usuario a través de su interfaz gráfica. El usuario puede valorar los datos que seleccione en la gráfica. Una vez realizada la valoración, introduce los estados de validación en la base de datos junto a sus datos correspondientes.
- **Programa de visualización:** Es un programa diseñado para ser ejecutado desde un navegador Web. Esta aplicación lee los datos de la base de datos y los muestra en una gráfica en el navegador.
- **Servicio de publicación Web:** Publica el programa de visualización en la red. El departamento informático de Confederación Hidrográfica del Guadiana es el encargado de hacerlo accesible desde Internet.
- **Formulario Access**: Ofrece al técnico los datos de la base de datos en un formato apto para la realización de los informes necesarios.

1.5 Estructura de la Memoria.

La memoria de este proyecto se compone de cuatro capítulos y cuatro anexos.

En el primer capítulo se realiza una presentación del proyecto. Se exponen los objetivos que tiene que cumplir, una explicación de la situación precedente a la aplicación de esta solución y el motivo por el que se lleva a cabo. Finalmente se describe la arquitectura de la solución y se resume la estructura de la memoria.

El segundo capítulo habla del entorno de desarrollo y las herramientas usadas. Se explica toda la tecnología utilizada en el proyecto y se expone la motivación de su uso.

El desarrollo de la solución es mostrado en el tercer capítulo. Está dividido en las partes de las que consta el trabajo realizado.

En el cuarto capítulo se presentan las conclusiones del proyecto y se comenta los trabajos siguientes que han sido realizados gracias a lo aprendido en éste.

El primer anexo muestra la configuración de Microsoft SQL Server 2014.

En el segundo anexo se expone la programación de tareas para la copia de los archivos de recuperación en los otros equipos

El tercer anexo contiene el proceso de configuración de SYSMAC Gateway.

Por último, en el cuarto anexo se explica la instalación del servicio Web utilizado (IIS en Windows Server 2008 R2).

2 TECNOLOGÍAS UTILIZADAS

La elección del entorno de desarrollo y de las herramientas de trabajo es de gran importancia a la hora de realizar un proyecto por diferentes motivos. En primer lugar, algunas tecnologías permiten al desarrollador aplicar ciertas soluciones que con otras no estarían disponibles. No sólo eso, la elección de una herramienta conocida por el desarrollador hace que éste trabaje de forma más rápida y eficaz. En cambio, la utilización de una nueva herramienta, aunque requiere de un periodo de aprendizaje, aporta nuevos conocimientos y experiencias dándole así al profesional un abanico de posibilidades más amplio para futuros proyectos. Por último, se debe tener en cuenta las necesidades del cliente y ver qué tecnologías se adaptan mejor a ellas.

A continuación se exponen las herramientas utilizadas en la realización del proyecto y se explica el motivo de su selección.

2.1 Windows Server 2008 R2

Figura 2-1. Windows Server 2008 R2

Windows Server 2008 R2 **(Figura 2-1)** es un sistema operativo producido por Microsoft. Fue lanzado como una versión mejorada del sistema operativo Windows Server 2008, corrigiendo errores y añadiendo nuevas características. Las mejoras incluyen nuevas funcionalidades para Active Directory, nueva virtualización y características de gestión, la versión 7.5 del Servidor Web Internet Information Services (IIS), arquitectura 64-bits o soporte para 256 procesadores lógicos.

La arquitectura del Kernel de Windows está basada en dos capas principales, el modo Kernel y el modo Usuario tal y como se puede apreciar en la figura **(Figura 2-2)**. El sistema utiliza cada modo para diferentes tipos de programa.

Cuando se ejecuta una aplicación, esta accede al modo Usuario. En este modo Windows crea un proceso específico para cada aplicación. Cada aplicación obtiene su propia dirección virtual privada. De esta forma, ninguna aplicación puede alterar los datos pertenecientes a otra y tampoco acceden al espacio virtual del sistema operativo. El modo Usuario tiene limitado el acceso al Hardware y las aplicaciones tienen que recurrir a Windows API para solicitar los servicios del sistema.

En el modo Kernel, el código ejecutado tiene acceso directo a todo el Hardware y toda la memoria del equipo. Todos los procesos comparten un mismo espacio virtual y se puede acceder a los espacios de direcciones de todas las aplicaciones del modo Usuario.

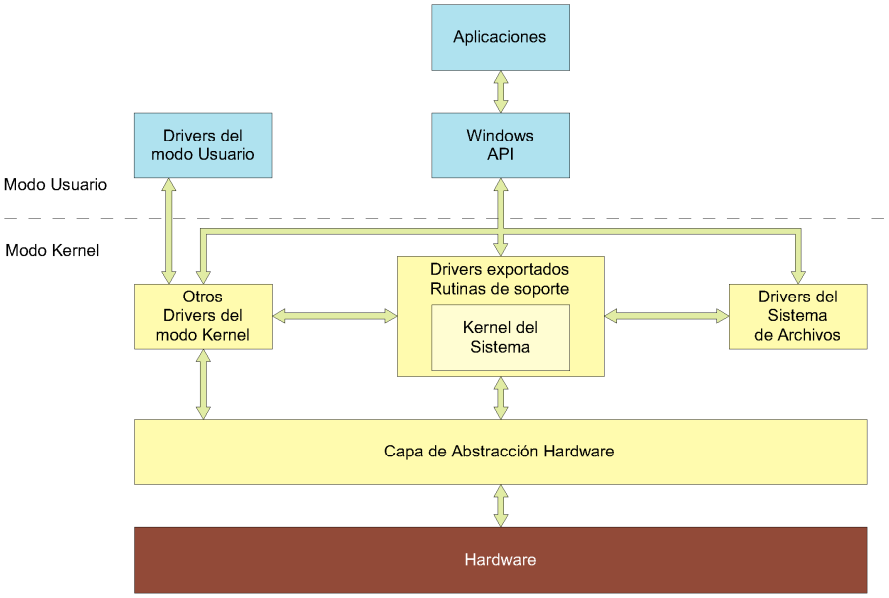

Figura 2-2. Arquitectura de Windows Server

Windows Server es una plataforma diseñada para compilar una infraestructura de aplicaciones, redes y servicios Web conectados desde el grupo de trabajo al centro de datos. Windows Server 2008 R2 permite al desarrollador mejorar sus aplicaciones mediante distintas funcionalidades disponibles en este sistema operativo. Algunos de los servicios más interesantes que ofrece Windows Server 2008 R2 son:

- **Servidor de aplicaciones:** Permite que un servidor hospede aplicaciones distribuidas creadas con ASP.NET, servicios empresariales y .NET Framework. Incluye más de una docena de servicios de función.
- **Servicios de archivos:** Son la base para compartir y replicar los archivos a través de la red y para la gestión de ellos. Incluye los servicios y subservicios Servidor de archivos, Sistema de archivos distribuido, Espacios de nombres DFS, Replicación DFS, Administrador de recursos del servidor de archivos, Servicios para Network File System, Servicio Búsqueda de Windows, Servicios de archivo de Windows Server 2003, Servicio de replicación de archivos y Servicios de Index Server.
- **Servicios de acceso y directivas de redes (NPAS):** Proporciona los servicios para la gestión del enrutamiento y el acceso remoto a redes. Incluye los servicios Servidor de directivas de redes, Servicios de enrutamiento y acceso remoto, Servicio de acceso remoto, Enrutamiento, Autoridad de registro de mantenimiento y Protocolo de autorización de credenciales de host.
- **Terminal Services:** Ofrece los servicios para la ejecución de aplicaciones basadas en Windows, instaladas en un servidor remoto. Incluye los servicios Terminal Server, Administrador de licencias TS, Agente de sesiones TS, Puerta de enlace TS y Acceso Web de TS.
- **Servicios Universal Description, Discovery, and Integration (UDDI):** Ofrece la capacidad de compartir información sobre los servicios Web en el interior de una organización y entre organizaciones. Incluye los servicios Base de datos de Servicios UDDI y Aplicación Web de Servicios UDDI.

- **Servidor Web (IIS):** Hospeda sitios y aplicaciones Web. Puede configurarse utilizando los módulos de IIS y las herramientas de administración. Incluye más de una docena de servicios de función.

Mediante el Administrador del Servidor de Windows Server 2008 R2 **(Figura 2-3)** se pueden configurar las funciones del servidor, los servicios de función y las características.

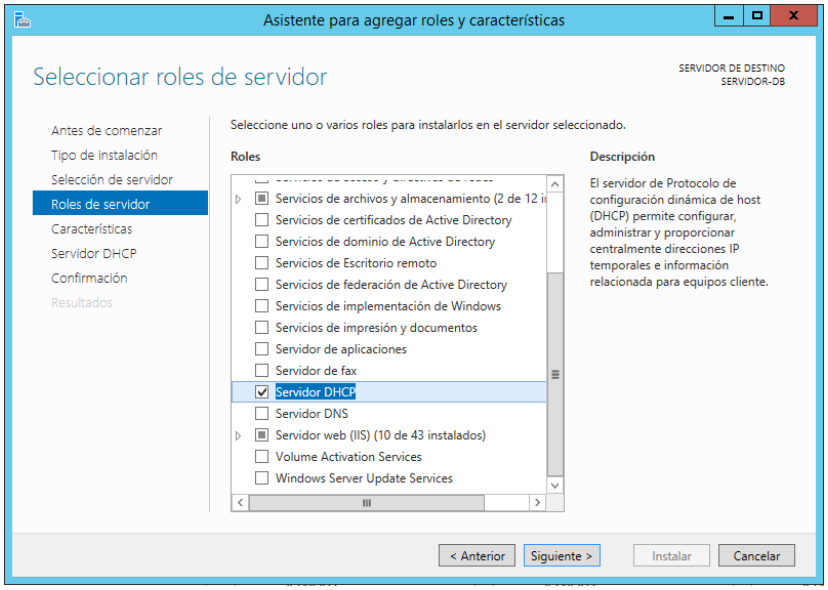

Figura 2-3. Administrador de roles de Windows Server 2008 R2

Las características que hacen interesante a este sistema operativo para este proyecto son las siguientes:

- Arquitectura 64 bits.
- La edición Standard permite hasta 32 GB de RAM.
- Dispone de .NET Framework 3.5 y .NET Framework 4.6.
- Ejecuta ASP.NET.
- Dispone de Internet Information Services (IIS).

2.1.1 Justificación

La motivación principal para elegir un sistema operativo de Microsoft sobre otros disponibles en el mercado es entregarle al cliente un entorno de trabajo conocido por ellos en el que no precisen de una adaptación. Una vez se ha tomado la decisión de optar por Windows, la elección de Windows Server 2008 R2 se debe a la necesidad de un sistema que cumpla con los siguientes requisitos:

- Proporcionar un servicio Web en el que se ejecuten aplicaciones Web.
- Ofrecer alta disponibilidad. El sistema debe estar en funcionamiento ininterrumpido.
- Poseer capacidad de soportar la gestión de una base de datos que irá creciendo con el transcurso del tiempo llegando a ser muy pesada.

El error cometido en los sistemas predecesores fue no predecir el crecimiento del tamaño de la base de datos. Esto condujo a la ralentización y mal funcionamiento a medida que la base de datos iba creciendo. Por lo tanto, la prioridad ha sido ofrecer un sistema estable bajo esas condiciones.

2.2 Microsoft SQL Server 2014

Figura 2-4. MS SQL Server 2014

Microsoft SQL Server 2014 **(Figura 2-4)** es un sistema de gestión de bases de datos relacionales diseñado por Microsoft. SQL Server permite almacenar, modificar y extraer información de una base de datos y proporciona herramientas para añadir, borrar, modificar y analizar datos. También proporciona métodos para asegurar la integridad de los datos, administrar el acceso y recuperar la información en caso de corrupción del sistema. Los procesos de manipulación de una base de datos por parte de SQL Server se muestran en el siguiente esquema **(Figura 2-5)**:

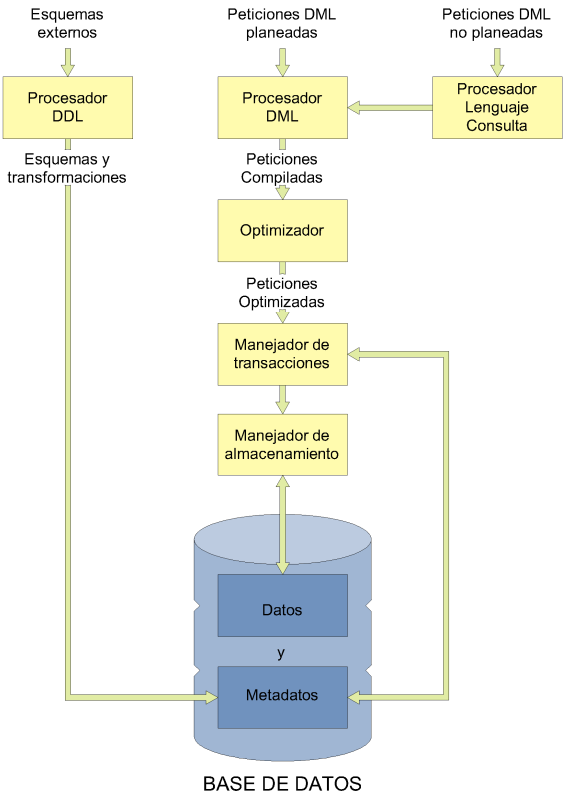

Figura 2-5. Esquema de un sistema gestor de bases de datos

El lenguaje que utiliza es SQL (Structured Query Language o Lenguaje de consultas estructuradas). SQL utiliza cuatro clases de sentencias. Las sentencias DDL (Data Definition Language), que se utilizan para crear, modificar o eliminar la estructura de una base de datos; las sentencias DML (Data Manipulation Language), que permiten recuperar, insertar, almacenar o modificar datos; las sentencias DCL (Data Control Language), que sirven para controlar el acceso a los elementos de una base de datos; y las sentencias TCL (Transactional Control Language), que permite administrar las transacciones que ocurren en el motor de bases de datos.

Una transacción de base de datos es un conjunto de sentencias SQL que se ejecutan como una secuencia de operaciones, formando una unidad lógica única de trabajo. Las transacciones deben ser ejecutadas completas o, de lo contrario, ninguno de los componentes de la transacción es ejecutado. Todas las transacciones de base de datos deben ser atómicas, coherentes, aisladas y duraderas.

SQL Server utiliza un conjunto de restricciones para conseguir la integridad de los datos. Esto incluye claves principales, claves foráneas, restricciones "No NULL", restricciones "UNIQUE", restricciones "Default" y restricciones "Check". Estas restricciones ayudan a garantizar la precisión y fiabilidad de los datos.

Microsoft SQL Server 2014 utiliza el lenguaje SQL como las versiones anteriores, pero tiene varios cambios significativos con respecto a su predecesor como mejoras en la copia de seguridad y restauración, conmutación e indexado de particiones, tablas con optimización para memoria, mejoras de implementación, mejoras de seguridad o mejoras de Transact-SQL.

Las características de SQL Server 2014 más destacadas son las siguientes:

- **Incluye un entorno gráfico de administración (Microsoft SQL Server Management Studio) (Figura 2-6)**. SQL Management Studio es la consola de administración principal de SQL Server. Permite crear, modificar o eliminar objetos de bases de datos (bases de datos, tablas, datos, vistas, procesos almacenados, etc.), visualizar los datos existentes en una base de datos, configurar cuentas de usuarios, realizar copias de seguridad, replicas o transferencias de datos entre bases de datos y gestionar todo lo posible en lo referente a bases de datos.

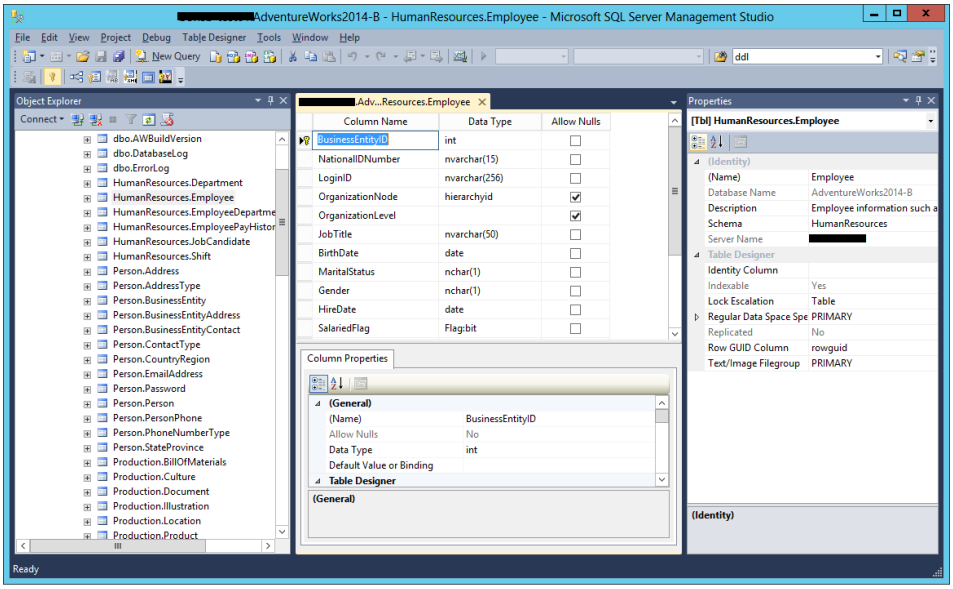

Figura 2-6. Microsoft SQL Server 2014 Management Studio

- **Soporta transacciones**. En SQL Server existen las instrucciones COMMIT o ROLLBACK que comunica al motor de bases de datos que si existe algún error en la secuencia de instrucciones de una transacción cancele todo lo que se haya ejecutado dentro de ésta.
- **Arquitectura cliente-servidor**. La comunicación de SQL Server con las aplicaciones que quieren utilizar las bases de datos utilizan la arquitectura cliente-servidor.
- **Soporta procedimientos almacenados**. Un procedimiento almacenado es un grupo de una o varias instrucciones Transact-SQL o una referencia a un método CLR de Microsoft .NET Framework. Con los procedimientos almacenados se consigue reducir el tráfico entre cliente-servidor, mayor seguridad en la ejecución, mejorar el rendimiento de la base de datos y un mantenimiento más sencillo.
- **Compresión de tablas**. Reduce el tamaño de las bases de datos mejorando la capacidad del sistema y el aprovechamiento de los recursos.
- **Respaldos y recuperaciones**. SQL Server permite la realización de copias de seguridad de bases de datos o partes de ellas, así como su restauración a un estado anterior. De esta manera, permite recuperar datos que puedan verse perdidos por motivos tales como corrupción de sistemas o errores humanos.
- **Alta disponibilidad**. Una solución de alta disponibilidad enmascara los efectos de un error de hardware o software y mantiene la disponibilidad de las aplicaciones a fin de minimizar el tiempo de inactividad que perciben los usuarios.

- **Programación de tareas**. A través de SQL Server Management Studio se puede programar tareas puntuales o periódicas. Las tareas llevadas a cabo pueden ser de tipo administrativas, de seguridad o de ejecución de sentencias.
- **Transact-SQL**.

2.2.1 Transact-SQL

Transact-SQL es el lenguaje de desarrollo utilizado por Microsoft SQL Server. Es una versión modificada del estándar ANSI del lenguaje SQL. Es utilizado para extraer datos, manipularlos, crear, modificar y eliminar tablas o definir relaciones entre ellas.

Transact-SQL es el medio más utilizado para la comunicación con el motor de la base de datos. Permite realizar operaciones en SQL Server y administrar el servidor. La interacción se realiza mediante envío de sentencias Transact-SQL y declaraciones que son procesadas por el motor y se devuelven los resultados al programa cliente.

SQL es un lenguaje orientado únicamente a la definición y al acceso a los datos por lo que no se puede considerar como un lenguaje de programación como tal ya que no incluye funcionalidades como son estructuras condicionales, bucles, formateo de salida, etc. Transact-SQL amplía el lenguaje SQL, ya que incluye características propias de cualquier lenguaje de programación, características que nos permiten definir la lógica necesaria para el tratamiento de la información:

- Tipos de datos.
- Definición de variables.
- Estructuras de control de flujo.
- Gestión de excepciones.
- Funciones predefinidas.

No obstante, no permite crear interfaces de usuarios ni crear aplicaciones ejecutables, sino elementos que llegan al servidor y son ejecutados.

Debido a estas restricciones, se utiliza para generar procedimientos almacenados, triggers y funciones de usuario.

2.2.2 Justificación

En un entorno de desarrollo bajo el sistema operativo Windows Server 2008 R2 existen varios sistemas gestores de bases de datos posibles. Las características de este proyecto invitan a elegir un gestor con capacidad para manejar bases de datos de gran tamaño. SQL Server 2014 cumple con estas condiciones. Además, aporta otras características que otros sistemas no ofrecen:

- Permite recuperar datos ante grandes desastres
- Cuenta con opciones de seguridad que le convierte en el gestor menos vulnerable
- No necesita la instalación de nada adicional para la comunicación con aplicaciones clientes en sistemas Windows
- Ofrece escalabilidad y rendimiento.
- Trabaja con transacciones permitiendo mantener coherencia en los datos.

2.3 Visual Studio 2013 Professional

Figura 2-7. Visual Studio 2013

Visual Studio **(Figura 2-7)** es un entorno de desarrollo integrado producido por Microsoft. Un entorno de desarrollo integrado es una aplicación informática que proporciona servicios integrales para ayudar al desarrollador en la creación de software.

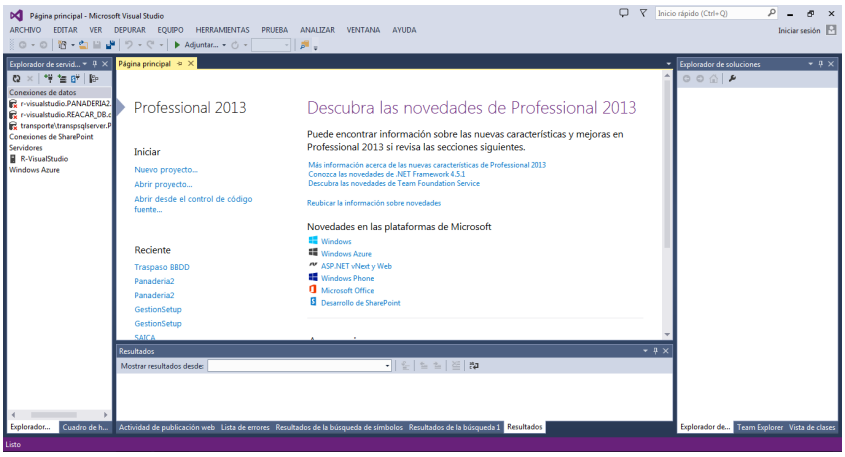

Figura 2-8. Interfaz de Microsoft Visual Studio 2013

Visual Studio 2013 cuenta con múltiples lenguajes de programación, como C++, C#, F#, Java, PHP, Python, Ruby y Visual Basic .NET. Además, ofrece un conjunto de herramientas que comprende todos y cada uno de los aspectos que están relacionados con la mayoría de los escenarios sobre los que puede realizarse programación de aplicaciones informáticas:

- **Editor de código con Intellisense predictivo y herramientas de refactorización y aceleración de código (Figura 2-9)**. El editor de código de Visual Studio ofrece al usuario herramientas para desarrollar el código de forma rápida, estructurada y limpia. Las herramientas de refactorización estructuran el código para que éste sea más claro. Intellisense sugiere al programador funciones, variables y otros tipos de elementos para no tener que escribir todo el texto. La aceleración de código completa acciones como cerrar corchetes, paréntesis o comillas para mantener el código coherente.

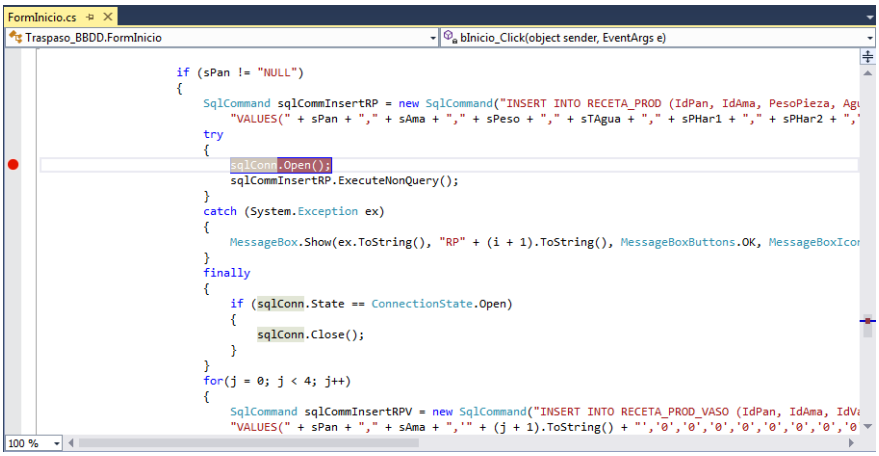

Figura 2-9. Editor de texto de Microsoft Visual Studio 2013

- **Análisis de código en estático**. Visual Studio analiza el código mientras el desarrollador está programando y señala los errores de compilación que va encontrando.
- **Test de aplicaciones**. Incluye una estructura de test unitario y de integración con una gestión de planes de pruebas.
- **Pruebas de cargas y rendimiento**. Las pruebas de carga permiten determinar y validar la respuesta de la aplicación cuando es sometida a una carga de usuarios o transacciones que se espera en el ambiente de trabajo. Las pruebas de rendimiento se realizan para medir la respuesta de la aplicación a distintos volúmenes de carga esperados.
- **Entorno de desarrollo Web**. Permite desarrollar aplicaciones Web en distintas plataformas (ASP.NET, MVC, Django...).
- **Construcción y depuración**. Construye aplicaciones para todos los dispositivos, plataformas y sistemas operativos. Realiza diferentes operaciones y validaciones de depuración permitiendo encontrar fallos fácilmente.
- **Conexiones con bases de datos**. Permite realizar conexiones con bases de datos SQL, comparar esquemas, comparar datos y lanzar consultas.
- **Creación de GUID**. Permite identificar la aplicación con un identificador global único. Esto permite al sistema dar un espacio de memoria único para esta aplicación.
- **Configuración de servicios WCF**. Windows Communication Foundation (WCF) es un marco para generar aplicaciones orientadas a servicios. Con WCF, es posible enviar datos como mensajes asincrónicos de un extremo de servicio a otro.
- **Conexión con servidores**. Visual Studio se conecta directamente con cualquier servidor, permitiendo al desarrollador obtener los recursos que los servidores ofrecen directamente desde el entorno de desarrollo.
- **Permite incluir herramientas de terceras partes**. Al ser Visual Studio en gran parte abierto y extensible, ofrece la posibilidad de incluir herramientas desarrolladas por terceras partes. Gracias a ello Visual Studio ofrece mayor funcionalidad.
- **Diseñador de formularios (Figura 2-10):** El diseñador de formularios proporciona muchas herramientas para compilar aplicaciones. Permite organizar controles utilizando líneas de ajuste, utilizar etiquetas inteligentes, establecer márgenes y rellenos para los controles o establecer los valores de propiedad de los controles mediante la ventana propiedad.

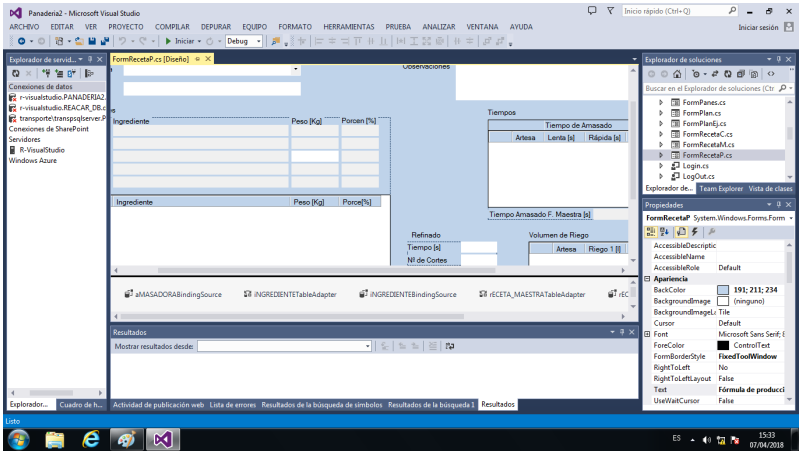

Figura 2-10. Diseñador de formularios de Windows de Visual Studio 2013

2.3.1 Justificación

La elección de este entorno de desarrollo integrado viene motivada por recomendaciones de compañeros del sector. Para la empresa ha resultado ser una inversión acertada apostar por Visual Studio.

Es una aplicación intuitiva, fácil de usar, posee una gran cantidad de librerías para numerosos usos, su curva de aprendizaje es rápida, tiene soporte de ayuda que cuenta con profesionales de Microsoft, goza de muchísima potencia y permite crear toda clase de aplicaciones.

En este proyecto resulta atractiva su capacidad de crear conexiones con bases de datos, la integración del lenguaje Transact-SQL y la posibilidad de incluir las herramientas necesarias para la conexión con los PLC de las estaciones de medidas.

2.4 .NET Framework

Figura 2-11. .NET Framework

.NET Framework **(Figura2-11)** es una tecnología que permite la compilación y ejecución de aplicaciones y servicios Web XML. El diseño de .NET Framework tiene las siguientes características:

- Proporciona un entorno coherente de programación orientada a objetos.
- Proporciona un entorno de ejecución de código que reduce la implementación de software y los conflictos de versiones.
- Promueve la ejecución segura del código, incluso del creado por terceras personas desconocidas o que no son de plena confianza.
- Elimina los problemas de rendimiento de los entornos en los que se utilizan scripts o intérpretes de comandos.
- Ofrece al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en Web.
- Libera los recursos utilizados cuando los procesos finalizan.
- Basa toda la comunicación en estándares del sector para asegurar que el código de .NET Framework se pueda integrar con otros tipos de código.

.NET Framework se compone de dos partes principales: Common Language Runtime (CLR) y la biblioteca de clases de .NET Framework.

Common Language Runtime se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la comunicación remota; al tiempo que aplica una seguridad de tipos estricta y otras formas de especificación del código que promueven su seguridad y solidez. El código destinado al tiempo de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado.

La biblioteca de clases es una colección completa orientada a objetos de tipos reutilizables que se pueden emplear para desarrollar aplicaciones que abarcan desde las tradicionales herramientas de interfaz gráfica de usuario (GUI) o de línea de comandos hasta las aplicaciones basadas en las innovaciones más recientes proporcionadas por ASP.NET, como formularios Web Forms y Servicios Web XML.

En la figura siguiente **(Figura 2-12)** se muestra la relación de Common Language Runtime y la biblioteca de clases con las aplicaciones y el sistema en su conjunto. En la ilustración se representa igualmente cómo funciona el código administrado dentro de una arquitectura mayor.

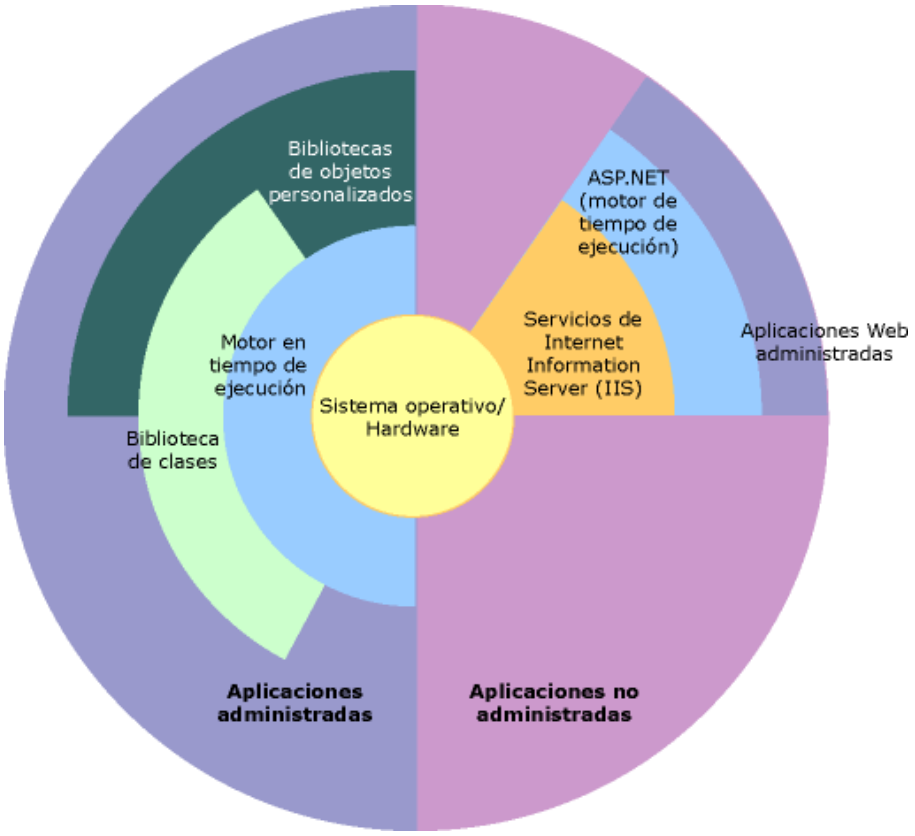

Figura 2-12. Arquitectura de trabajo de .NET Framework

2.4.1 Justificación

La utilización de .NET Framework está incluida en las aplicaciones generadas por Visual Studio. Aunque su uso no se debe a una elección, las características que posee durante el tiempo de ejecución son muy interesantes para el desarrollador de aplicaciones. Con ellas ayuda al programador a centrarse en el funcionamiento del programa puesto que .NET Framework se encarga de la ejecución segura de éste.

2.5 C#

Figura 2-13. Lenguaje de programación C#

C# **(Figuran2-13)** es un lenguaje de programación orientado a objetos que permite al desarrollador crear aplicaciones seguras y robustas ejecutadas en .NET Framework. Con C# se pueden crear aplicaciones clientes de Windows, componentes distribuidos, servicios Web XML, aplicaciones cliente-servidor, aplicaciones de base de datos entre otras muchas cosas.

La sintaxis de C# es altamente descriptiva, aunque simple y fácil de aprender. Pertenece a la familia de sintaxis de C, C++ y Java. La sintaxis de C# simplifica las complejidades de C++ y posee valiosas características como tipos de datos null, enumeraciones, delegaciones, expresiones lambda y accesos directos a memoria que no se encuentran en Java.

C# también incluye compatibilidad para programación orientada a componentes. Estos componentes presentan un modelo de programación con propiedades, métodos y eventos. También ofrecen atributos que da información sobre el componente y agrega su propia documentación.

Varias características de C# ayudan a la construcción de aplicaciones sólidas y duraderas:

- **La recolección de elementos no utilizados** automáticamente reclama la memoria ocupada por objetos no utilizados y no accesibles.
- **El control de excepciones** proporciona un enfoque estructurado y extensible para la detección de errores y la recuperación
- El diseño **del lenguaje con seguridad de tipos** hace imposible leer desde las variables sin inicializar, indexar matrices más allá de sus límites o realizar conversiones de tipos no comprobados.

C# tiene un sistema de tipo unificado. Todos los tipos de C# se heredan de un único tipo raíz. Todos los tipos comparten un conjunto de operaciones comunes, y los valores de todos los tipos se pueden almacenar, transportar y utilizar de manera coherente. Además, C# admite tipos de valor y tipos de referencia definidos por el usuario, lo que permite la asignación dinámica de objetos.

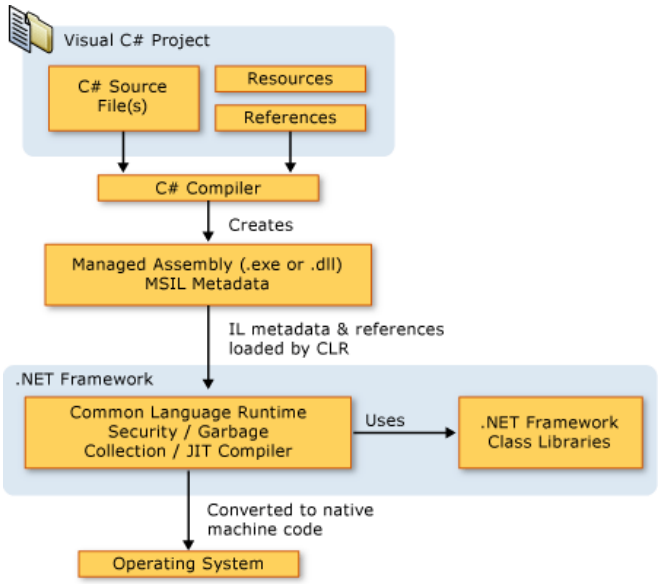

Figura 2-14. Arquitectura de .NET Framework

Un programa desarrollado en C# corre en .NET Framework, un componente integral de Windows que incluye un sistema de ejecución virtual (CLR) y un conjunto de librerías de clases unificadas. El código fuente de C# es compilado en un lenguaje intermedio (IL). El código IL y los recursos como strings o mapas de bits son almacenados en disco junto con un archivo ejecutable llamado ensamblado. Cuando un programa C# es ejecutado, el ensamblado es cargado en el CLR. Entonces, si cumple los requisitos de seguridad, el CLR realiza una compilación y convierte el código IL en lenguaje nativo de la máquina. Todo este proceso viene representado en la figura anterior **(Figura 2-14)**.

Las principales características del lenguaje de programación C# son:

- **Lenguaje de alto nivel**. C# es un lenguaje altamente descriptivo y comprensible, válido para diversas máquinas que permite crear programas complejos con mucho menos líneas que en otro tipo de lenguajes.
- **Orientado a objetos**. Soporta objetos, que son abstracciones de datos con una interfaz de operaciones con nombre y un estado local oculto. Los objetos tienen un tipo asociado (clase). Las clases pueden heredar atributos de los supertipos (superclases).
- **Orientado a componentes**. Un componente es un paquete de software, un servicio Web o un módulo que encapsula un conjunto de funciones relacionadas. Un sistema orientado a componentes coloca todos los procesos en componentes separados. Los componentes se comunican entre ellos a través de interfaces. Un componente puede ser sustituido por otro mientras el segundo cumpla los mismos

requisitos que el primero. La programación orientada a componentes permite la actualización en tiempo de diseño o en tiempo de ejecución de parte del código y la reutilización de componentes en otros programas. Esto ofrece como beneficio reducir costes de desarrollo y mantenimiento.

- **Recolección de basura**. Tiene incluido el recolector de basura del CLR de .NET Framework que se encarga de la destrucción de objetos cuando no son necesarios. Así, el programador no está obligado a añadir en su código las instrucciones de destrucción.
- **Seguridad de tipos**. El lenguaje provee de una serie de normas de sintaxis para que no se produzcan errores difíciles de detectar.
- **Instrucciones seguras**. Para evitar errores muy comunes, en C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes.
- **Unificación de tipos**. Todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada System.Object. El hecho de que todos los tipos del lenguaje deriven de una clase común facilita enormemente el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.
- **Extensión de tipos básicos**. Permite definir tipos de datos para los que se apliquen las mismas optimizaciones que para los tipos de datos básicos.
- **Extensión de operadores básicos**. Permite redefinir el significado de la mayoría de los operadores cuando se apliquen a diferentes tipos de objetos.

2.5.1 Justificación

Como otras decisiones tomadas en este proyecto, la motivación principal es que el sistema que se desarrolla se va a ejecutar en un entorno Windows. C# y Java son dos lenguajes de programación parecidos con el suficiente bagaje para que en ambos se puedan desarrollar las mismas aplicaciones. La principal ventaja de C# sobre Java en este proyecto es que C# se adapta mejor a su ejecución en Windows. La solución, además, contiene un programa desarrollado en el entorno ASP.NET que requiere que parte de su programación sea en Visual Basic o C#, por lo tanto, para unificar la mayor parte posible del código del proyecto, se ha preferido usar C#.

C# es un lenguaje fácil de aprender cuando se posee cierta base de conocimientos de programación. Posee características muy útiles para el desarrollo y la ejecución de aplicaciones como son la seguridad de tipos y la recolección de basura. Además, como permite realizar todo tipo de programas, en un proyecto con tanta funcionalidades como éste ofrece los recursos necesarios para su desarrollo.

En resumen, los motivos para usar C# son la ejecución en un sistema operativo Windows y el trabajo en el entorno de desarrollo Web ASP.NET.

2.6 ASP.NET

Figura 2-15. ASP.NET

ASP.NET **(Figura 2-15)** es un entorno para aplicaciones Web creado por Microsoft. Permite el desarrollo de sitios Web y aplicaciones con HTML, CSS y JavaScript. También puede crear API Web y usar tecnologías en tiempo real.

Como muestra la figura siguiente **(Figura 2-16)**, un usuario accede a una aplicación ASP.NET a través del servicio dado por IIS. Una vez iniciada la aplicación, .NET Framework se encarga de la ejecución del código.

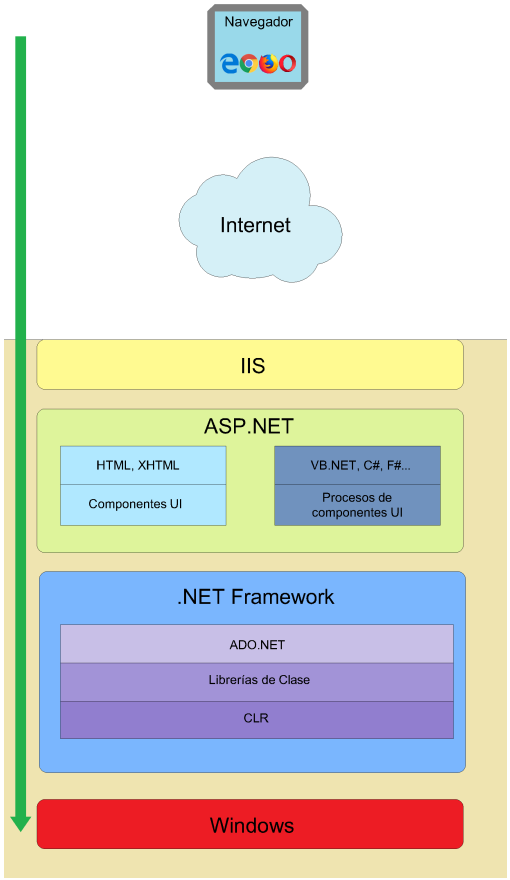

Figura 2-16. Arquitectura de ejecución de una aplicación ASP.NET

Las páginas de ASP.NET son el principal medio de construcción. Los formularios Web están diseñados con etiquetas HTML o XHTML estático, etiquetas que definen Controles Web que se procesan en el servidor y Controles de Usuario que son creados por los desarrolladores con el código estático y dinámico que requieran.

Para la programación dinámica se recomienda el uso del modelo code-behind. Este modelo consiste en diseñar la estética del formulario Web en un archivo que incluye el código estático, y almacenar el código dinámico en un archivo separado del diseño. Esta práctica se realiza automáticamente en Visual Studio. Para la programación del código dinámico puede usarse Visual Basic .NET o C#.

ASP.NET permite la creación de componentes reutilizables. Éstos son los Controles de Usuario, que son controles diseñados por el desarrollador donde puede incluir propiedades, métodos y manejadores de eventos. Los Controles de Usuario pueden disparar eventos en el formulario Web de ASP.NET.

Cuando se ejecuta una página ASP.NET, ésta recorre un ciclo de vida en el que realiza una serie de pasos de procesamiento. En términos generales, la página recorre las fases descritas a continuación:

- **Solicitud de página**. La solicitud de página se produce antes de que comience el ciclo de vida de la página. Cuando un usuario solicita la página, ASP.NET determina si ésta se debe analizar y compilar o si se puede enviar una versión en caché de la página como respuesta sin ejecutar la página.
- **Inicio**. En el paso de inicio, se establecen las propiedades de la página, como Request y Response. En esta fase, la página también determina si la solicitud es una devolución de datos o una nueva solicitud, y establece la propiedad IsPostBack. Además, durante esta fase se establece la propiedad UICulture de la página.
- **Inicialización de página**. Durante la inicialización de la página, los controles incluidos en ella están disponibles y se establece la propiedad UniqueID de cada uno de ellos. Además, se aplican los temas correspondientes a la página. Si la solicitud actual es una devolución de datos, los datos de devolución

aún no se han cargado y los valores de las propiedades del control no se han restaurado a los valores del estado de vista.

- **Carga**. Durante la carga, si la solicitud actual es una devolución de datos, las propiedades del control se cargan con información recuperada del estado de vista y del estado del control.
- **Validación**. Durante la validación, se llama al método Validate de todos los controles de validación, que establece la propiedad IsValid de cada uno de los controles de validación y de la página.
- **Control de eventos de devolución de datos**. Si la solicitud es una devolución de datos, se llama a los controladores de eventos.
- **Representación**. Durante la representación, el estado de vista se guarda en la página y, a continuación, ésta llama a cada uno de los controles para que aporte su salida representada al valor OutputStream de la propiedad Response de la página.
- **Descarga**. Se llama a la descarga cuando la página se ha representado completamente, se ha enviado al cliente y está lista para ser descartada. Llegado este momento, se descargan las propiedades de la página, como Response y Request, y se llevan a cabo las operaciones de limpieza correspondientes.

Todo este proceso se representa en el siguiente esquema **(Figura 2-17)**:

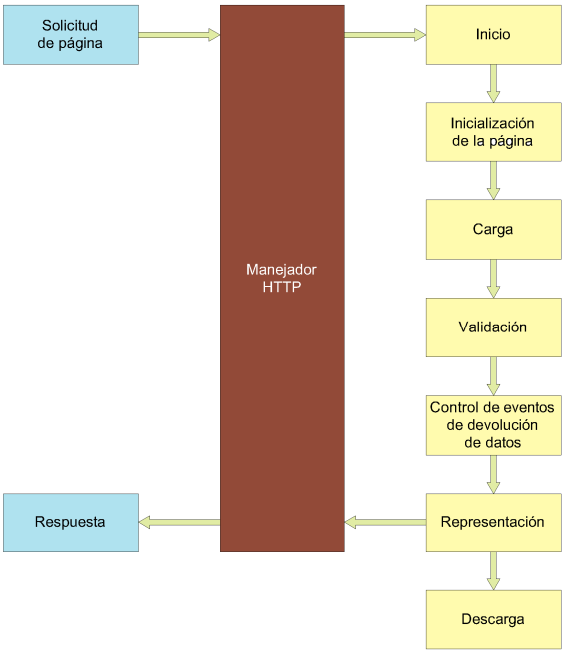

Figura 2-17. Ciclo de vida de un programa ASP.NET

En resumen, las características más reseñables que ofrece ASP.NET son:

- Separación del código dinámico y del diseño.
- Diseño en HTML o XHTML estático. Las
- Programación dinámica en Visual Basic .NET, C# o F#.
- Uso de Controles Web.
- Creación de Controles de Usuario.
- Administración del estado.
- Motor de plantillas.

2.6.1 Justificación

La elección de ASP.NET como tecnología para el desarrollo de la aplicación Web de este proyecto se debe a las diferentes ventajas que ofrece frente a otras opciones.

Una de las ventajas de ASP.NET con respecto a otras tecnologías de programación de aplicaciones Web es que la aplicación se compila en vez de leerse secuencialmente. Esto conlleva un aumento de la velocidad de respuesta y rendimiento en el servidor y su ejecución es más segura y robusta.

El código dinámico está separado del diseño de la interfaz de usuario. Esto facilita la programación ya que se pueden realizar cambios en el diseño Web sin afectar al funcionamiento de la aplicación o lo contrario, modificar el funcionamiento si afectar el diseño.

La creación de una aplicación en ASP.NET se puede realizar desde Visual Studio, que es un entorno de desarrollo que ayuda enormemente a la programación por las razones detalladas anteriormente.

2.7 CX-Compolet y SYSMAC Gateway

Figura 2-18 CX-Compolet + SYSMAC Gateway

SYSMAC Gateway es un software de comunicaciones desarrollado por OMRON. Sólo es ejecutable en Windows. Admite enlaces de comunicaciones CIP (Common Industrial Protocol) y nombres de etiquetas (Ethernet/IP), además de funciones diseñadas por OMRON. SYSMAC Gateway configura las comunicaciones entre un ordenador y un dispositivo controlador, por ejemplo, un PLC o un variador. Permite crear puertos de comunicación por el que una aplicación puede conectarse a un dispositivo remoto que utilice el protocolo CIP o los datos de etiquetas.

Las ventajas que SYSMAC Gateway ofrece en la comunicación entre ordenadores y PLCs se exponen a continuación:

- Las operaciones de SYSMAC Gateway han sido verificadas bajo Ethernet/IP.
- Ofrece una memoria virtual de PLC que permite a un PC participar como nodo de enlace de datos.
- Los cambios en memoria pueden ser detectados en aplicaciones ejecutadas en un PC.
- El estado de SYSMAC Gateway puede ser comprobado en una bandeja de tareas.
- Permite reducir el tiempo de programación al absorber las diferencias entre redes.
- Permite enlaces de datos veloces y de gran capacidad sin necesidad de instalar tarjetas PCI especiales.
- La eficiencia de las comunicaciones puede ser optimizada.

CX-Compolet **(Figura 2-18)** es un conjunto de componentes de software necesarios para la comunicación entre un PC y controladores de OMRON. Este paquete incluye librerías y controles .NET y ActiveX que pueden utilizarse con los lenguajes Visual Basic.NET y C#. Además, admite la comunicación mediante el uso de nombres de etiquetas Ethernet/IP con las familias de controladores CJ2 y NJ/NX. También acepta tipos de datos como estructuras y matrices. El paquete es instalable en Visual Studio para trabajar con él.

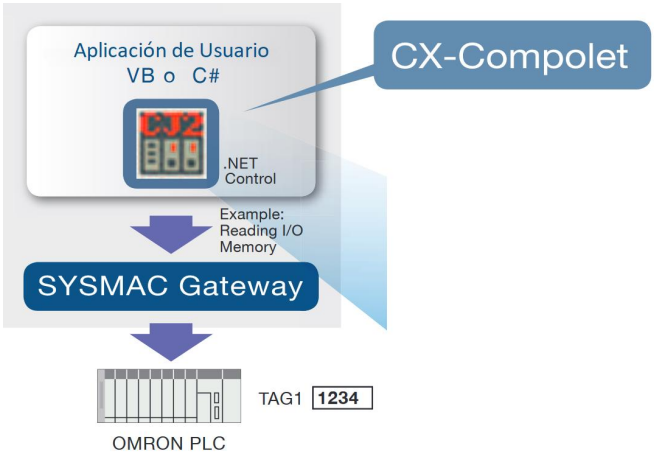

Figura 2-19. Funcionamiento de CX-Compolet y SYSMAC Gateway

La asociación de estas dos tecnologías permite el desarrollo de aplicaciones con capacidad de comunicarse con dispositivos de control que utilicen el protocolo CIP o el sistema de nombre de etiquetas Ethernet/IP. Las funciones que ofrecen las librerías de CX-Compolet utilizan los enlaces de comunicación creados por SYSMAC Gateway. SYSMAC Gateway trabaja como driver de comunicaciones entre el PC y los dispositivos de OMRON. Este programa traduce las peticiones de las funciones de CX-Compolet y realiza la comunicación con los PLCs. Este funcionamiento aparece reflejado en la anterior figura **(Figura 2-19)**.

2.7.1 Justificación

El modelo de los PLC que se encuentran en las estaciones de medida es de la familia de controladores CJ2 de OMRON. Para realizar la comunicación con ellos, OMRON ofrece varias soluciones. Entre ellas se encuentran el paquete CX-Compolet y SYSMAC Gateway.

Aunque es la solución que requiere más programación por parte del usuario, es la que posee mayor potencia porque permite desarrollar funciones más complejas a partir de las funciones básicas que ofrece. Para el desarrollo de aplicaciones grandes, es la solución que OMRON recomienda. Y un factor que ayuda también a decantarse por esta opción, es que es más económica que otras soluciones más sencillas de utilizar, pero que no poseen tanto potencial.

2.8 Internet Information Services (IIS) de Windows Server 2008 R2

Figura 2-20. Internet Information Services (IIS)

Internet Information Services (IIS) **(Figura 2-20)** es un servidor Web para el sistema operativo Windows. El rol de Servidor Web en Windows Server 2008 R2 (IIS 7.5) ofrece un servicio seguro, fácil de administrar, modular y extensible donde hospedar sitios Web, servicios y aplicaciones.

Los componentes de IIS están divididos en dos grandes grupos: un grupo de componentes que se encuentra en el modo Kernel y un grupo de componentes que están incorporados en procesos de modo usuario.

Los componentes en modo Kernel están ubicados por encima de la capa del protocolo TCP/IP y se comunican directamente con el driver TCPIP.sys. Dentro del modo Kernel se encuentra el driver HTTP.sys que es el driver HTTP para IIS. El propósito de este driver no es más que el de escuchar todo el tráfico HTTP a través del HTTP Listener, interpretarlo y asegurarse de que este tráfico es válido. Una vez que se ha verificado la

validez del tráfico, HTTP.sys coloca las peticiones en el Request Queue correspondiente para que un Worker Process la tome para su procesamiento. El driver HTTP.sys realiza la decodificación SSL y Windows Authentication se maneja a través de la caché de HTTP.sys.

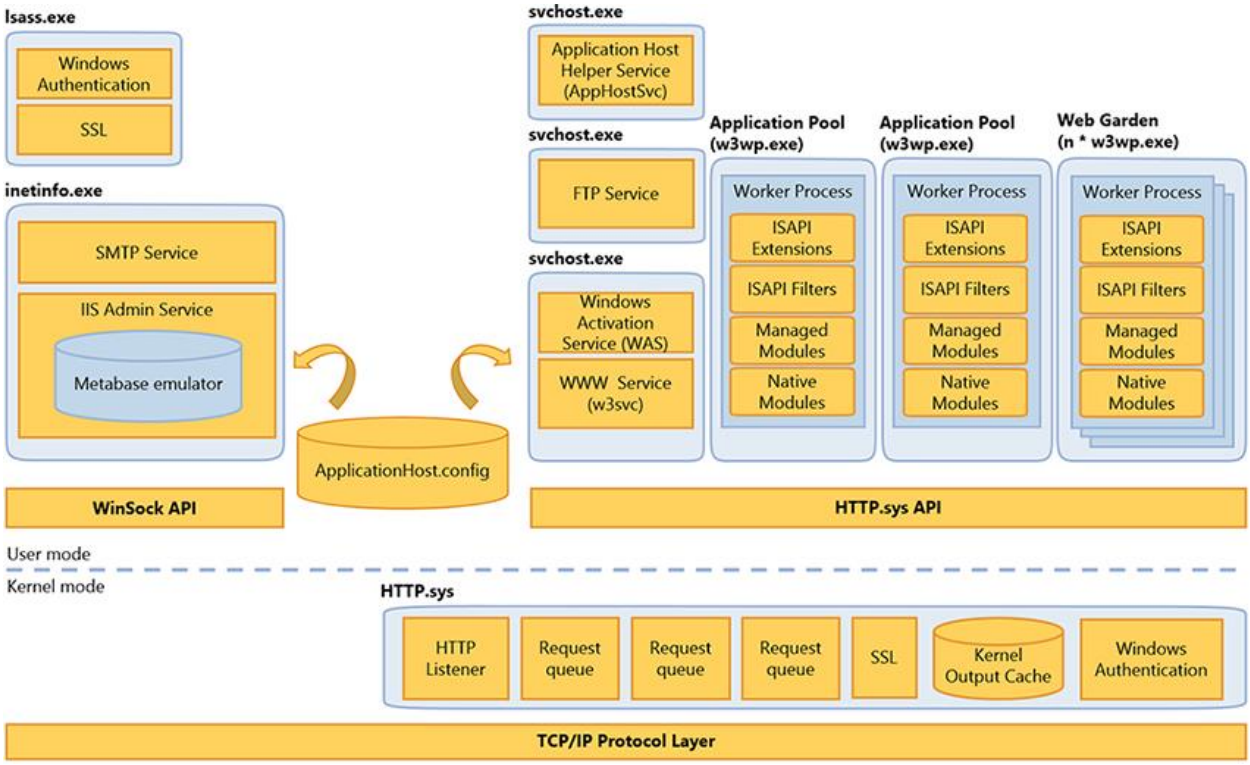

Figura 2-21. Arquitectura de IIS 7.5

Uno de los componentes del modo usuario presente en la arquitectura es inetinfo.exe. Dentro de este proceso se ejecutan el Servicio SMTP y el Servicio de Administración de IIS, cuyo propósito es administrar y cargar la Metabase de IIS en memoria. Los Application Pools son una agrupación de URLs enrutadas a un Worker Process. Un Worker Process es el proceso de Windows responsable de procesar las peticiones Web. También existen los Web Gardens que son Application Pools que utilizan más de un Worker Process en un sólo servidor. En el modo usuario también se encuentran los procesos svchost.exe, encargados de realizar las tareas de iniciar y manejar los Worker Process y permitir trabajar con protocolos distintos a HTTP (WAS); dar Servicio WWW (w3svc); dar Servicio FTP; y realizar tareas de respaldo de la configuración de IIS presente en el archivo ApplicationHost.config (AppHostSvc). Isass.exe es utilizado para la Decodificación SSL y Windows Authentication, con el agregado de que una parte importante de su implementación reside en HTTP.sys

Cuando se inicia un servidor que corre IIS, el Servicio WAS lee la información del archivo de configuración ApplicationHost.config y crea el Setup para HTTP.sys de manera similar como sucedía con el Servicio WWW al iniciar la tabla de enrutamiento.

HTTP.sys valida la petición que recibe:

- Si la petición es inválida: se envía el error HTTP correspondiente al cliente.
- Si la petición es válida, HTTP.sys verifica la posibilidad de dar respuesta a la petición desde su cache.
    - Si la respuesta existe en el cache, HTTP.sys la envía inmediatamente
    - De lo contrario, HTTP.sys coloca la petición en la cola de peticiones específica para el Application Pool correspondiente según la tabla de enrutamiento.

Seguidamente HTTP.sys pregunta si existe un Worker Process que pueda procesar la petición:

- Si existe uno o más Worker Process ejecutándose y escuchando la cola de peticiones, el Worker Process toma la petición directamente de la cola, la procesa y posteriormente envía la respuesta a HTTP.sys y al cliente.

- De no ser así, HTTP.sys le Noifica al Servicio WWW que se necesita un Worker Process. A su vez, el Servicio WWW pide al Servicio WAS que inicie y configure un Worker Process basado en la información alojada en el archivo ApplicationHost.config.

IIS ofrece gran cantidad de funcionalidades. Las más útiles para la realización de este proyecto son:

- **Compilación de .NET**. Administra la compilación del código de las aplicaciones ASP.NET.
- **Perfil de .NET**. Proporciona la administración de las propiedades de perfiles que se utilizan para realizar el seguimiento de la información personalizada requerida por la aplicación.
- **Niveles de confianza .NET**. Configura el nivel de seguridad de acceso del código (CAS) que se otorga a una aplicación.
- **Grupo de aplicaciones**. Puede administrar un conjunto de aplicaciones en un servidor Web.
- **Configuración de aplicaciones**. Realiza la administración de una lista de pares clave-valor que se almacena en el archivo Web.config de la aplicación.
- **Aplicaciones**. Administra las aplicaciones de un sitio Web.
- **Compresión**. Ofrece un tiempo de transmisión más rápido entre IIS y los exploradores habilitados para la compresión.
- **Examen de directorios**. Modifica la configuración del contenido que se puede ver en un directorio del servidor Web.
- **Redirección HTTP**. Configura el modo en que las solicitudes entrantes se redirigen a un nuevo destino.
- **Encabezados de respuesta HTTP**. Controla una lista de pares de nombre y valor que contienen información sobre una página solicitada y para configurar encabezados HTTP comunes.

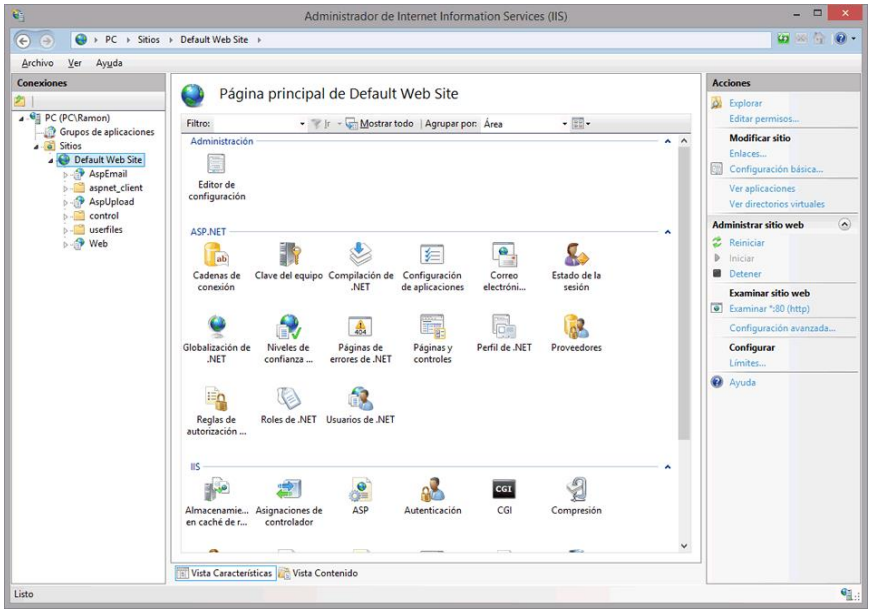

Figura 2-22. Administrador de IIS

- **Administrador de IIS (Figura 2-22)**. Representa los componentes de la interfaz de usuario que están disponibles en el nivel superior del Administrador de IIS.
- **Registro**. Configurar cómo registra IIS las solicitudes realizadas al servidor Web y cuándo se crean nuevos archivos de registro.
- **Clave del equipo**. Modifica las opciones de algoritmos hash y cifrado usadas por los servicios de la aplicación, como el estado de vista, la autenticación mediante formularios, la pertenencia y los roles, y la identificación anónima.

- **Servicio de administración**. Ofrece el servicio de administración que permite a los administradores de equipos y dominios administrar de forma remota un servidor Web que utiliza el Administrador de IIS.
- **Páginas y controles**. Permite configurar las opciones de las páginas y controles ASP.NET.
- **Estado de sesión**. Define el comportamiento de la información mantenida en las sesiones del explorador.
- **Sitios**. Administra los sitios en un servidor Web.

IIS 7.5 está compuesto por los módulos mostrados en la siguiente imagen **(Figura 2-23)**.

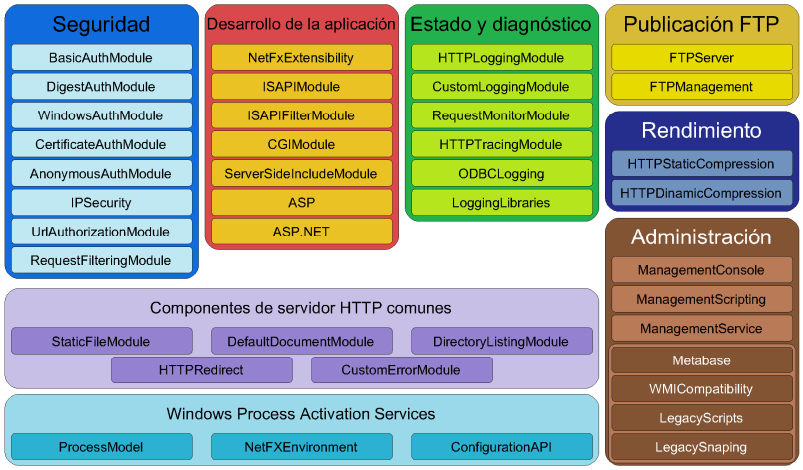

Figura 2-23. Módulos de componentes de IIS

2.8.1 Justificación

IIS es el servidor Web que proporciona de manera predeterminada Windows Server 2008 R2. Para ello sólo es necesario activar el rol de servidor Web y estará disponible para su uso. De esta forma no es necesario instalar ningún otro servidor Web ya que proporciona todo lo necesario para hospedar la aplicación Web desarrollada en esta solución.

Gracias a su administrador, permite gestionar de forma rápida y sencilla el funcionamiento de un sitio Web y las aplicaciones que se ejecutan en él.

2.9 Access de Microsoft Office 2013

Figura 2-24. Microsoft Access

Microsoft Access **(Figura 2-24)** es un gestor de bases de datos incluido dentro del conjunto de aplicaciones de Microsoft Office. Con Access se puede agregar datos a una base de datos, modificar datos existentes en la base de datos, eliminar información, organizar y mostrar los datos de diferentes formas o compartir los datos con otras personas mediante informes, correos electrónicos, intranet o Internet.

Una base de datos Access se constituye de las siguientes partes:

- **Tablas**. Una tabla de Access se organiza en filas y columnas. Las filas almacenan información y las columnas definen la información almacenada en esas filas.

- **Formularios**. Los formularios permiten crear una interfaz de usuario en la que controlar los datos de una base de datos. En los formularios se pueden realizar acciones para la gestión de datos, generación de informes, trasvase de datos o ejecutar otras tareas.
- **Informes**. Los informes se usan para dar formato a los datos, resumirlos y presentarlos.
- **Consultas**. Las consultas realizan diferentes funciones en una base de datos. Existen dos variedades de consultas: consulta de selección y consulta de acciones. Una consulta de selección extrae los datos que se especifiquen de la base de datos y los pone a disposición para su uso. Las consultas de acción se utilizan para crear tablas, agregar datos, modificarlos o eliminarlos.
- **Macros**. Las macros se utilizan en Access para agregar funciones a la base de datos. Las macros se utilizan como secuencias de acciones para automatizar el control de la base de datos por parte del usuario permitiendo ahorrar mucho tiempo.
- **Módulos**. Los módulos son objetos que se pueden utilizar para añadir funcionalidad a la base de datos. Los módulos se escriben en lenguaje Visual Basic para Aplicaciones (VBA). Un módulo es una colección de declaraciones, instrucciones y procedimientos que se almacenan juntos como una unidad.

Además, Access permite la conexión a otros tipos de base de datos como bases de datos SQL mediante ODBC y gestionarlas utilizando las partes de Access descritas.

2.9.1 Justificación

El gestor de base de datos principal que se usa en este proyecto es SQL Server 2014. Debido a que SQL Server no es una herramienta que un técnico no especializado en bases de datos sepa manejar fácilmente, se ha utilizado Microsoft Access para algunas funciones. Access permite crear un formulario para que un usuario pueda extraer los datos que precise de la base de datos de forma sencilla y sin posibilidad de modificarla o corromperla. Ya que el técnico responsable del centro de control tiene conocimientos básicos sobre los programas contenidos en Microsoft Office, puede realizar informes con Access o traspasar los datos a otros formatos como por ejemplo Excel.

3 DESARROLLO DE LA SOLUCIÓN

El propósito principal de este proyecto es desarrollar un sistema de recopilación y almacenamiento de datos estable, duradero y automático para el control de la calidad de aguas superficiales y residuales en la cuenca del río Guadiana. Además, se ha de proveer al cliente de herramientas útiles para la inspección de los datos recopilados y su publicación.

El punto de partida es un conjunto de sistemas desfasados y en malfuncionamiento que no son modificables por ninguna empresa que no haya participado en sus desarrollos. En consecuencia, se ha de crear un sistema más moderno que pueda utilizar los recursos aprovechables de los sistemas anteriores

3.1 Diseño del sistema.

El sistema desarrollado está formado físicamente por 29 estaciones de toma de datos de la red SAICA, 16 estaciones de medidas de la red REACAR y un centro de control localizado en Badajoz. Todas las estaciones del sistema y el centro de control se encuentran dentro de la red de comunicaciones de Confederación Hidrográfica del Guadiana como se muestra en la figura **(Figura 3-1)**. Esto quiere decir que los distintos puntos del sistema son completamente accesibles desde dentro de dicha red.

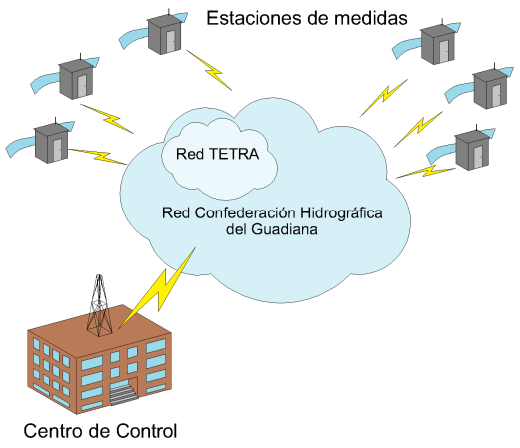

Figura 3-1. Esquema de la red de comunicación del sistema

Las estaciones de medida de la red SAICA y las estaciones de la red REACAR no son exactamente iguales. Mientras SAICA se encarga de la toma de datos en aguas superficiales, REACAR adquiere los datos de aguas residuales que son vertidas a las aguas superficiales. Por consiguiente, los aparatos de medidas que utilizan no son los mismos en las dos redes debido a que no se necesita tomar las mismas variables. De hecho, entre las distintas estaciones de la red SAICA puede haber diferencias entre los datos tomados.

Una estación SAICA completamente equipada está compuesta por una red de sensores, un PLC de OMRON, una pantalla HMI para el control local de la estación, un switch de comunicaciones, un router 3G y un módulo radio/módem de red TETRA conectados tal como muestra el esquema **(Figura 3-2)**. La red de sensores está conectada al PLC y se compone de un dispositivo AquaTest de ADASA que toma las medidas del pH, temperatura del agua, turbidez, oxígeno disuelto, conductividad y redox (reacción de reducción-oxidación); un sistema ADI2004 de Applikon que mide los niveles de amonio, nitrato y fosfato; un sistema SC1000 de HACH que capta la cantidad de potasio, cloruro y carbono orgánico; y sondas analógicas para la temperatura dentro de la estación, el nivel de las aguas y el caudal. El módulo radio/módem está conectado al PLC por cable serie y se comunica con la red de comunicaciones del cliente. El PLC, a su vez, está conectado al switch por cable Ethernet junto a la pantalla HMI y al router 3G que realiza un túnel VPN hacia la red de comunicaciones.

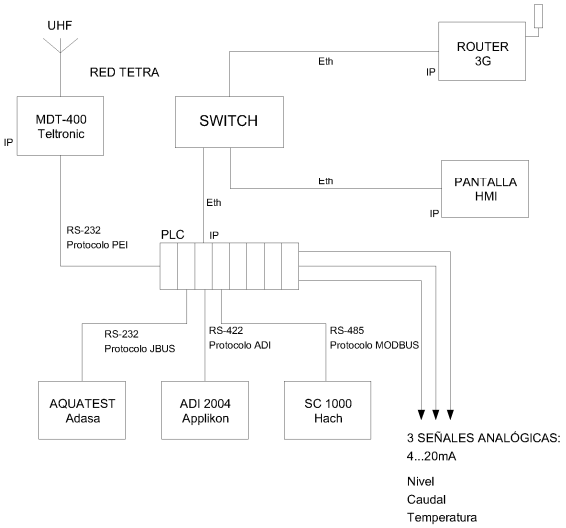

Figura 3-2. Infraestructura de una estación de la red SAICA

Por otra parte, las estaciones REACAR poseen un dispositivo AquaTest de ADASA como red de sensores, un PLC de OMRON, una pantalla HMI para el control local, un switch, una cámara IP y un router 3G. Al igual que en las estaciones SAICA, el router 3G tiene un túnel VPN que conecta con la red de comunicaciones del cliente. El AquaTest está conectado con el PLC y el PLC al switch junto a la pantalla HMI, la cámara IP y el router. El esquema de una estación REACAR queda como muestra la figura **(Figura 3-3)**.

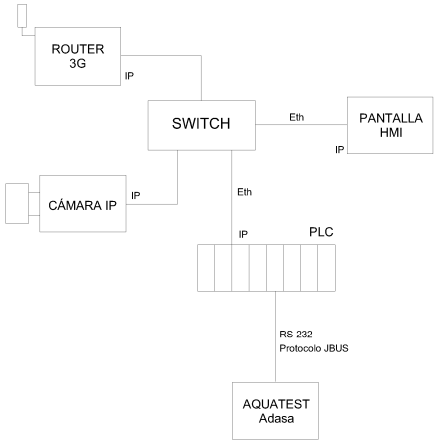

Figura 3-3. Infraestructura de una estación de la red REACAR

En el centro de control se hallan dos servidores con un procesador de 8 núcleos a 2GHz, 32GB de RAM, 1TB de capacidad de disco duro y una tarjeta PCI de comunicaciones Ethernet. Ambos servidores se encuentran enganchados por cable a la red de comunicaciones de Confederación Hidrográfica del Guadiana como se ve en la figura **(Figura 3-4)**. Cada servidor se encarga de una de las redes de seguimientos. Es decir, un servidor está dedicado para la red SAICA y el otro servidor controla la red REACAR.

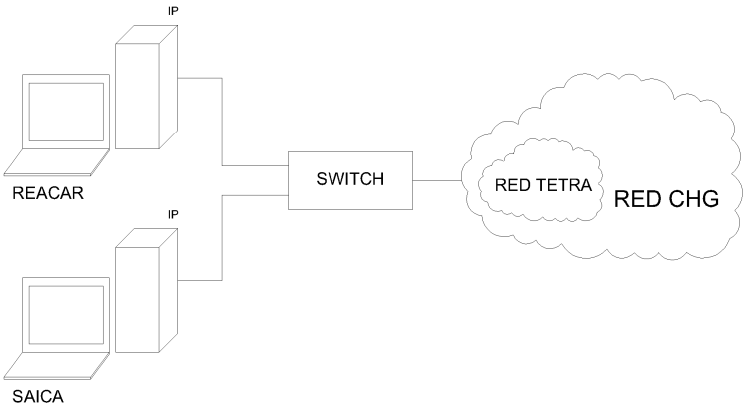

Figura 3-4. Infraestructura del centro de control

Los objetivos a cumplir por el sistema son los expuestos en el **Capítulo 1**. Se exponen de nuevo a continuación:

- La adquisición de datos por parte del sistema debe ser totalmente automática. El usuario sólo ha de introducir en el programa las direcciones IP de las estaciones de medidas. Las direcciones IP serán almacenadas y podrán ser modificadas por el usuario cuando sea necesario.
- Una vez realizada la lectura de datos de una estación, los datos serán almacenados en una base de datos para su posterior tratamiento.
- La aplicación mostrará los datos almacenados al usuario en una gráfica de tendencia.
- El usuario podrá validar los datos directamente en la gráfica.
- La interfaz gráfica debe ser fácil de usar y visualmente agradable para que no canse al técnico que trabaje en la validación de datos y realización de informes.
- El usuario podrá elegir qué variables quiere validar, seleccionar el periodo de tiempo a valorar y dar un concepto de validación a los valores pertenecientes a la selección realizada.
- La gráfica de tendencia debe ser navegable. Debe tener la opción de representar desde 1 hora a 1 mes completo y avanzar y retroceder en el tiempo de manera sencilla. También se podrá ir a una fecha concreta de manera directa.
- El usuario podrá seleccionar qué estaciones de medidas están operativas. Las estaciones de medida se irán modificando una vez se haya desarrollado la solución e irán estando a disposición del nuevo sistema gradualmente.
- Desde Internet se tendrá acceso a datos válidos almacenados en la base de datos a través de una publicación Web para su consulta.
- La publicación Web sólo ofrecerá los datos válidos de la semana anterior a la semana en que se realiza la consulta.
- Los datos de la publicación Web mostrará los datos en una gráfica de líneas de tendencia.
- El técnico del centro de control podrá elegir qué datos se publican.
- Se debe desarrollar una herramienta para que el técnico pueda extraer los datos que necesite de la base de datos en un formato que pueda trabajar para realizar informes sin que exista la posibilidad de alterar los datos almacenados.

Una vez se ha estudiado la finalidad que debe alcanzar el sistema desarrollado, los procesos que ha de cometer son los definidos a continuación:

- **Proceso en las estaciones de medida (Figura 3-5)**. Este proceso se realiza en todas las estaciones de medida, ya sean de la red SAICA o REACAR. Es un proceso cíclico que es ejecutado por el PLC cada 15 minutos. Los pasos de los que se compone son los siguientes:
    - Toma de datos. El PLC lee los datos de la red de sensores.
    - Almacenamiento en memoria interna del PLC. El PLC guarda los datos obtenidos en registros distribuidos de manera ordenada y conocida por el centro de control para su posterior lectura. Si la memoria está completa, el PLC sobrescribe los registros de la toma de datos más antigua.
    - Lectura desde el centro de control. Una vez almacenados los datos de medida, el PLC espera a que el centro de control se conecte y recopile la información. Cuando el centro de control consigue extraer todos los registros disponibles, envía al PLC una confirmación de lectura finalizada. Mientras el PLC no recibe esta confirmación, sigue esperando hasta que se cumpla el tiempo de ciclo y vuelva a ejecutar el primer paso.
    - Borrado de datos. Todos los registros almacenados son eliminados si el PLC recibe la confirmación de lectura por parte del centro de control.

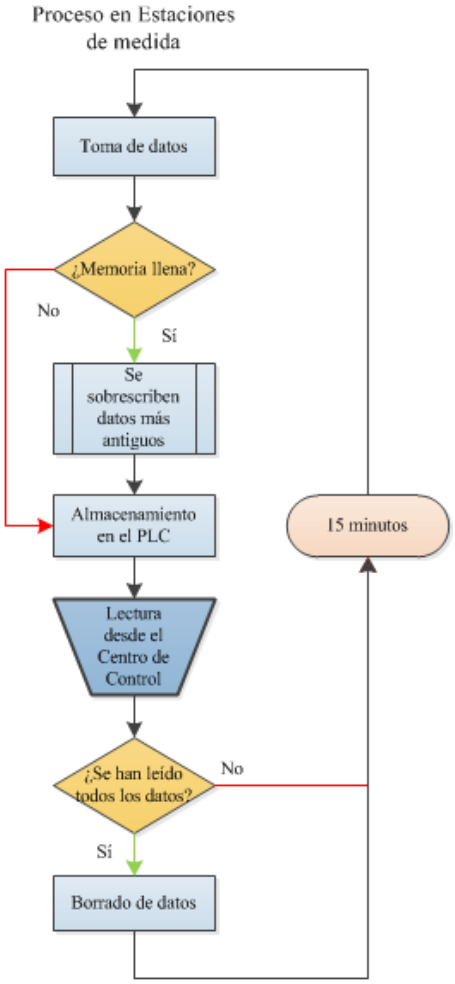

Figura 3-5. Proceso de las estaciones de medidas

- **Proceso de lectura desde el centro de control (Figura 3-6)**. La aplicación de validación ejecuta el proceso de lectura. Para cada estación se crea un proceso en un hilo independiente. El proceso es ejecutado cada 15 minutos para cada estación. El proceso de lectura para la red SAICA es distinto del proceso de lectura para la red REACAR. Esto se debe a la redundancia de redes que existe en la red

SAICA, excluyendo del proceso de la red REACAR el reintento de conexión por otra vía. El proceso de lectura de la red SAICA se compone de los siguientes pasos:

- Conexión con estación: La aplicación intenta conectar con la estación SAICA mediante el método de conexión preferente. Si no se consigue realizar la conexión, intenta conectarse con la estación utilizando el otro método disponible. Si tampoco se establece la conexión, la aplicación cierra el hilo del proceso y espera a que se inicie el ciclo de nuevo. En el proceso de lectura de la red REACAR, la conexión se intenta por la única vía posible. En el caso de no poderse conectar con la estación, el proceso finaliza.
- Lectura del PLC: Una vez se obtiene comunicación con la estación de medida, la aplicación extrae todos los datos disponibles en el PLC.
- Almacenamiento en Base de datos: La aplicación ordena los datos obtenidos y los almacena en las tablas correspondientes dentro de la base de datos.
- Confirmación al PLC: Cuando se ha terminado los procesos de lectura del PLC y escritura en la base de datos, la aplicación envía una confirmación al PLC para que éste sepa que el proceso se ha completado satisfactoriamente.

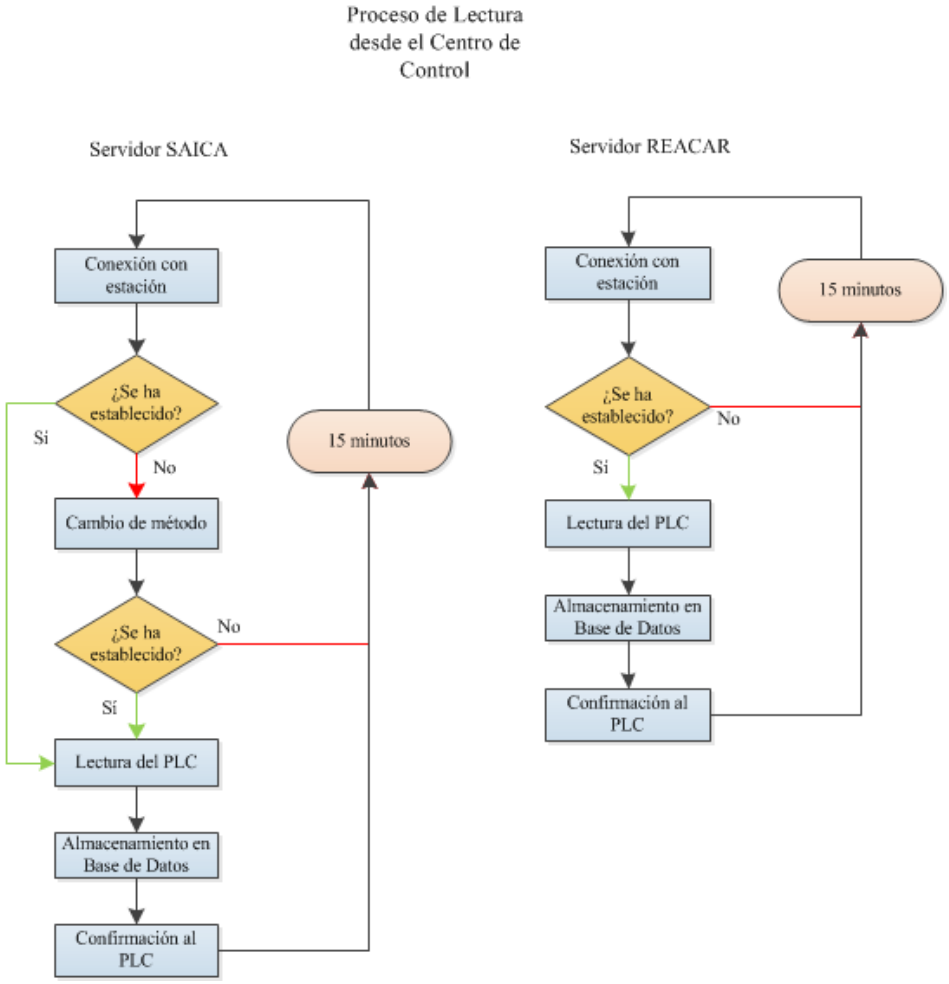

Figura 3-6. Procesos de lectura desde el Centro de Control

- **Proceso de validación (Figura 3-7)**. El proceso de validación es llevado a cabo por la aplicación de validación. Se ejecuta en el hilo principal de la aplicación y necesita la participación del técnico del centro de control para realizarse.
  - Introducción de parámetros de búsqueda: El proceso de validación comienza cuando el técnico selecciona la estación que quiere revisar. Esta parte del proceso también incluye la introducción de las fechas a consultar y la selección de las variables a mostrar.

- Lectura de Base de datos: Cuando el técnico ha introducido cualquier valor de búsqueda en la aplicación, ésta realiza una consulta en la base de datos y extrae los datos solicitados.
- Representación gráfica: La aplicación muestra los datos obtenidos de la base de datos en una gráfica representando los valores de las variables y las fechas de lectura de esos valores.
- Validación: El técnico selecciona los datos sobre los que desea realizar la validación y les asigna un valor de validez.
- Modificación de la Base de datos: La aplicación modifica el estado de validez de cada dato seleccionado por el técnico en la base de datos.

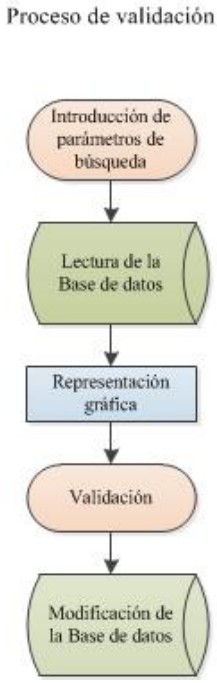

Figura 3-7. Proceso de validación

- **Proceso de publicación Web (Figura 3-8)**. Este proceso es realizado por la aplicación de visualización Web desarrollada con ASP.NET y el servidor Web IIS de Windows Server. Permite a un usuario consultar datos de la base de datos a través de la red.
  - Solicitud de página: El proceso comienza cuando un usuario introduce la url o dirección IP del servidor en un navegador Web. En este momento se solicita la página al servidor Web.
  - Envío de página principal: El servidor Web responde a la solicitud por parte del navegador Web del usuario enviando la página principal del programa de visualización.
  - Selección de estación: El usuario selecciona en la página principal qué estación quiere consultar. En este momento, el navegador solicita al servidor una nueva página con los datos de la estación seleccionada.
  - Lectura de Base de datos: La aplicación extrae de la base de datos los datos solicitados por el usuario.
  - Representación gráfica: Se representan los datos en una gráfica con los valores solicitados y la fecha de lectura de esos valores. Una vez finalizada la creación de la página, el servidor Web la envía y el navegador la muestra en pantalla.
  - Navegación: El usuario puede seleccionar en la página recibida qué variables quiere visualizar y desplazarse a través de la línea temporal de la gráfica. Cada vez que se realiza una selección en la página, el navegador Web insta al servidor a reenviar la página con las modificaciones solicitadas.

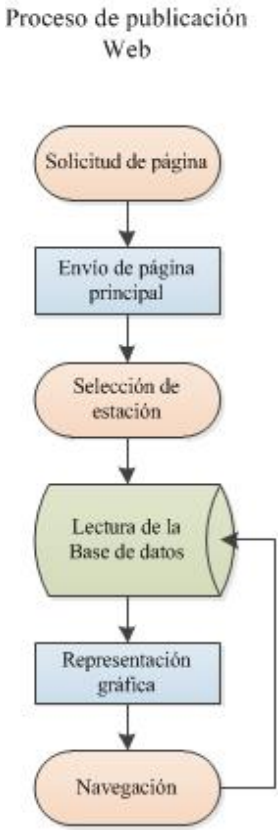

Figura 3-8. Proceso de publicación Web

- **Consulta de datos y generación de informes (Figura 3-9)**. A través de un formulario en Access, se permite al técnico del centro de control obtener los datos que precise de la base de datos SQL. Los datos se ofrecen en una tabla y puede realizar con ellos las acciones que se le antoje, puesto que en ningún caso afectaría a los datos almacenados en la base de datos SQL.
    - Introducción de parámetros de búsqueda: El técnico que trabaja en el centro de control introduce en un formulario los filtros necesarios para la búsqueda de los datos que quiere extraer. El formulario solicita al técnico la estación que se quiere consultar y el inicio y el fin de las fechas de lectura.
    - Lectura de Base de datos: El formulario realiza la consulta al servidor SQL con los parámetros introducidos.
    - Creación de tablas de datos: Una vez obtenidos los datos, el formulario crea una tabla en Access en la que introduce estos valores y se la muestra al técnico. Una vez aquí, el técnico tiene varias opciones:
    - Realización de informe: El técnico puede realizar un informe directamente en Access y entregarlo a quien se los haya solicitado.
    - Exportación de datos: Los datos se pueden exportar a otro tipo de programas ofimáticos, como por ejemplo, Excel.

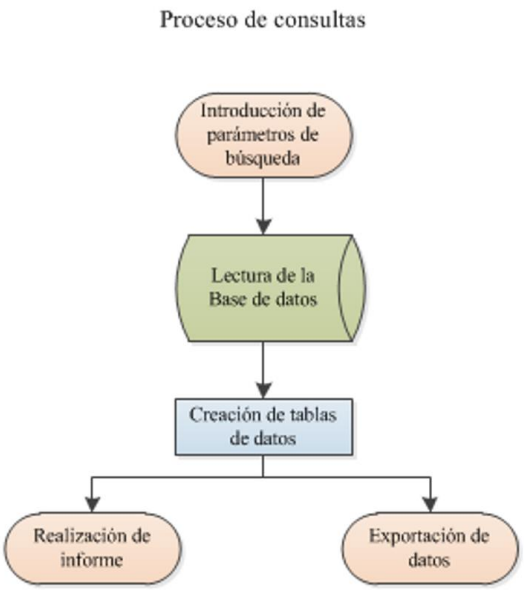

Figura 3-9. Proceso de consultas Access

4 BASE DE DATOS

Antes de comenzar cualquier proceso, es necesario la existencia de una base de datos donde guardar la información. En esta base de datos no sólo se almacenan los valores de las variables sobre la calidad de las aguas, también se recopila la información necesaria para el funcionamiento del sistema, como las direcciones IP de las estaciones, los nombres de las variables y sus unidades, datos de control de eventos o preferencias en el uso de las aplicaciones.

Las bases de datos de los sistemas de control de las redes SAICA y REACAR difieren por distintos motivos: número distintos de estaciones, número distinto de variables tomadas y modos de comunicación con las estaciones. Estas diferencias se ven reflejadas en las tablas ESTACIONES y VARIABLES, y en cada tabla donde se almacenan los datos leídos de cada estación de medida, explicándose cada cambio en su apartado correspondiente.

4.1 ESTACIONES

En la tabla estaciones se almacena información de cada estación de medida y datos útiles para la comunicación con ellas. Esta tabla es distinta para los dos sistemas de control. Los campos de la tabla ESTACIONES en la red SAICA se describen a continuación:

- **Id**: Representa el identificador de la estación. Es una cadena de caracteres de tamaño 3 (tipo de dato: *nchar(3)*). Es la llave principal de la tabla. No permite valores *NULL*.
- **Estaciones**: Este campo contiene el nombre identificativo de la estación. Es una cadena de caracteres de tamaño máximo 30 (*nvarchar(30)*). No permite dos valores iguales. No permite valores *NULL*.
- **Río**: Contiene el nombre del río donde se halla la estación. Es una cadena de caracteres de tamaño máximo 30 (*nvarchar(30)*).
- **Municipio**: Contiene el nombre del municipio donde se encuentra la estación. Es una cadena de caracteres de tamaño máximo 30 (*nvarchar(30)*).
- **IPVPN**: Este campo almacena la dirección IP del PLC por la que se accede a través de la conexión VPN entre la estación y la red de comunicaciones de Confederación. Es una cadena de caracteres de tamaño máximo 15 (*nvarchar(15)*).

- **IPTETRA**: Almacena la dirección IP del PLC por la que se accede a través de la conexión TETRA. Es una cadena de caracteres de tamaño máximo 15 (*nvarchar(15)*).
- **V_T**: Identifica la preferencia de método de conexión del programa de validación con el PLC. Es un tipo de dato booleano (*bit*) donde *true* significa que el método primario es mediante TETRA y *false* hace que el método primario sea mediante VPN. No permite valores *NULL*.
- **Activo**: Es una variable tipo bool (*bit*). Cuando es *true* quiere decir que la estación está activa y el programa lee los datos periódicamente del PLC, mientras que cuando es *false* el programa no realiza ningún intento de lectura de datos de dicha estación. No permite valores *NULL*.

En la figura **(Figura 4-1)** se muestra el diseño de la tabla en el Administrador de SQL Server.

Nombre de columna	Tipo de datos	Permitir val...
Id	nchar(3)	☐
Estaciones	nvarchar(40)	☐
Río	nvarchar(40)	☑
Municipio	nvarchar(40)	☑
IPVPN	nvarchar(15)	☑
IPTETRA	nvarchar(15)	☑
V_T	bit	☐
Activo	bit	☐

Figura 4-1. Tabla ESTACIONES de la base de datos de SAICA

Mientras que la tabla ESTACIONES de la base de datos de la red REACAR se compone de los siguientes campos:

- **Id**: Representa el identificador de la estación. Es una cadena de caracteres de tamaño 3 (tipo de dato: *nchar(3)*). Es la llave principal de la tabla. No permite valores *NULL*.
- **Estaciones**: Este campo contiene el nombre identificativo de la estación. Es una cadena de caracteres de tamaño máximo 30 (*nvarchar(30)*). No permite dos valores iguales. No permite valores *NULL*.
- **Localización**: Informa si la estación pertenece a la red *Oriental* o la red *Occidental*. Es una cadena de caracteres de tamaño máximo 15 (*nvarchar(15)*).
- **IPVPN**: Este campo almacena la dirección IP del PLC por la que se accede a través de la conexión VPN entre la estación y la red de comunicaciones de Confederación. Es una cadena de caracteres de tamaño máximo 15 (*nvarchar(15)*).
- **Activo**: Es una variable tipo bool (*bit*). Cuando es *true* quiere decir que la estación está activa y el programa lee los datos periódicamente del PLC, mientras que cuando es *false* el programa no realiza ningún intento de lectura de datos de dicha estación. No permite valores *NULL*.

Al no poseer comunicación mediante la red TETRA, la base de datos del sistema REACAR sólo almacena una dirección IP. Tampoco incluye el campo V_T porque éste identifica la preferencia del canal de comunicación entre TETRA y VPN y en este caso sólo existe una única posibilidad.

4.2 VARIABLES

En la tabla VARIABLES se definen las variables que podrán ser tomadas en las estaciones. Esta tabla es distinta para los dos sistemas de control. Los campos de la tabla VARIABLE para la red SAICA son los siguientes:

- **Id**: Representa el identificador de la variable. Es la llave principal de la tabla y el tipo de variable es un entero (*int*) autoincremental. No permite valores *NULL*.
- **Variables**: Contiene el nombre de la variable. Es una cadena de caracteres de tamaño máximo 40 (*nvarchar(40)*). No permite dos valores iguales. No permite valores *NULL*.

- **Ud_Ing**: Este campo contiene la unidad de ingeniería de la variable. Es una cadena de caracteres de tamaño máximo 10 (*nvarchar(10)*).
- **V_Max**: El valor almacenado en este campo indica el valor máximo representado en la gráfica de los programas de validación y visualización Web de la variable. Es una variable de tipo coma flotante (*real*). No permite valores *NULL*.
- **V_Min**: El valor almacenado en este campo indica el valor mínimo representado en la gráfica de los programas de validación y visualización Web de la variable. Es una variable de tipo coma flotante (*real*). No permite valores *NULL*.
- **S401, ... , S430**: Estos campos son de tipo binario (*bit*). Cuando su valor es *true* la variable es representada por defecto en la gráfica de la estación del programa de validación y además se puede mostrar en la gráfica de la estación en el programa de visualización Web. Cuando su valor es *false* no es posible representar la variable en la gráfica de la estación en el programa de visualización Web y, por defecto, la variable no se muestra en la gráfica del programa de validación. No permite valores *NULL*.

En la figura **(Figura 4-2)** se muestra el diseño de la tabla en el Administrador de SQL Server.

Nombre de columna	Tipo de datos	Permitir val...
Id	int	☐
Variable	nvarchar(40)	☐
Ud_Ing	nvarchar(10)	☑
V_Max	real	☐
V_Min	real	☐
S401	bit	☐
S402	bit	☐
S403	bit	☐
S404	bit	☐
S406	bit	☐
S407	bit	☐
S408	bit	☐
S409	bit	☐
S410	bit	☐
S411	bit	☐
S412	bit	☐
S413	bit	☐
S414	bit	☐
S415	bit	☐
S416	bit	☐
S417	bit	☐
S418	bit	☐
S419	bit	☐
S420	bit	☐
S421	bit	☐
S422	bit	☐
S423	bit	☐
S424	bit	☐
S425	bit	☐
S426	bit	☐
S427	bit	☐
S428	bit	☐
S429	bit	☐
S430	bit	☐

Figura 4-2. Tabla VARIABLES de la base de datos SAICA

La diferencia entre la tabla VARIABLE de la base de datos de la red SAICA y la tabla de la red REACAR viene dada por los campos referentes a las estaciones. En el caso de REACAR se definen de la siguiente forma:

- **R01, ... , R16**: La definición de estos campos es la misma que en el caso de la red SAICA. Son campos de tipo binario (*bit*). Cuando su valor es *true* la variable es representada por defecto en la gráfica de la estación del programa de validación y además se puede mostrar en la gráfica de la estación en el programa de visualización Web. Cuando su valor es *false* no es posible representar la variable en la gráfica de la estación en el programa de visualización Web y, por defecto, la variable no se muestra en la gráfica del programa de validación. No permite valores *NULL*.

4.3 VALIDEZ

La tabla VALIDEZ contiene los estados de validez que pueden tomar los datos de medidas almacenados en la base de datos. Esta tabla es igual en la base de datos de la red SAICA y en la de la red REACAR. Los campos que contiene la tabla VALIDEZ se definen a continuación:

- **Id**: Es el identificador del valor de validez. Es la llave principal de la tabla y el tipo de variable es un entero (*int*) autoincremental. No permite valores *NULL*.
- **Estado**: Indica si el dato al que se le relaciona es *Válido*, *No válido* o está *Sin validar*. Es una cadena de caracteres de tamaño máximo 11 (*nvarchar(11)*). No permite valores *NULL*.
- **Motivo**: Almacena el motivo por el que el valor al que se le relaciona tiene la validez dada. Es una cadena de caracteres de tamaño máximo 30 (*nvarchar(30)*). No permite valores *NULL*.
- **Forma**: Indica la forma del punto donde se representa el dato al que está relacionado. Por ejemplo, un aspa o un triángulo. Es una cadena de caracteres de tamaño máximo 10 (*nvarchar(10)*). No permite valores *NULL*.

En la figura **(Figura 4-3)** se muestra el diseño de la tabla en el Administrador de SQL Server.

Nombre de columna	Tipo de datos	Permitir val...
Id	int	☐
Estado	nvarchar(11)	☐
Motivo	nvarchar(30)	☐
Forma	nvarchar(10)	☐

Figura 4-3. Tabla VALIDEZ

4.4 S401 a S430 y R01 a R16

Las tablas donde se guardan los datos de lectura de las distintas variables sobre calidad de agua son de la S401 a la S430 para la red SAICA y de la R01 a la R16 para la red REACAR. Cada tabla corresponde a una estación diferente. Los campos de los que están compuestas son los siguientes:

- **Fecha_Hora**: Este campo contiene la hora y la fecha a la que se ha tomado la lectura de la red de sensores por parte del PLC. El tipo de variable que utiliza está diseñado para almacenar fecha y hora (*datetime*). Forma parte de la llave principal de la tabla. No permite valores *NULL*.
- **Variable**: El campo variable define de qué variable es el dato almacenado en el registro. Este campo es un entero (*int*) y está relacionado con el campo Id de la tabla VARIABLE. Junto con el campo Fecha_Hora son la llave principal de la tabla. No permite valores *NULL*.
- **Valor**: Contiene el valor del dato medido. Es una variable de tipo coma flotante (*real*). No permite valores *NULL*.
- **Validez**: Especifica la validez del dato contenido. Es un entero (*int*) y está relacionado con el campo Id de la tabla VALIDEZ. No permite valores *NULL*.

- **Porcentaje**: Contiene el valor del dato medido en porcentaje donde el 100% viene dado por el campo V_Max de la tabla VARIABLES y el 0% por V_Min. Es una variable de tipo coma flotante (*real*). No permite valores *NULL*.

Todas las tablas de las estaciones son iguales, tanto para la base de datos SAICA como para REACAR. Lo único que cambia es el nombre de la tabla.

En la figura **(Figura 4-4)** se muestra el diseño una de las tablas en el Administrador de SQL Server.

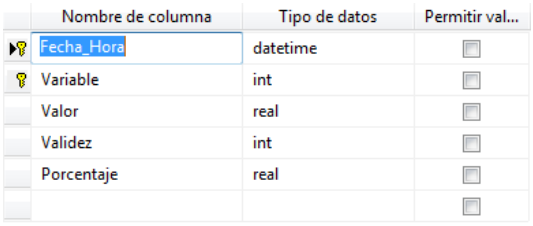

Figura 4-4. Tabla de estación S401 de la base de datos de SAICA

4.5 EVENTOS

La tabla EVENTOS almacena mensajes generados por eventos producidos durante la comunicación con las estaciones. Estos eventos pueden ser cambios de modo de conexión o errores de comunicación. La tabla es la misma en la base de datos SAICA y REACAR, aunque en la red REACAR no se va a producir eventos para cambiar el modo de conexión. Los campos que forma la tabla son definidos a continuación:

- **Id**: Es un índice que realiza la función de llave principal de la tabla. Es un entero (*int*) autoincremental. No permite valores *NULL*.
- **IdEstación**: Contiene el identificador de la estación con la que se ha producido la eventualidad. Es una cadena de caracteres de tamaño 3 (*nchar(3)*). Está relacionada con la tabla ESTACIONES mediante el campo Id de ella. No permite valores *NULL*.
- **Fecha_Hora**: Almacena la hora y la fecha en la que se produce el evento. El tipo de variable contenido es *datetime*. No permite valores *NULL*.
- **Evento**: Contiene el mensaje producido por el evento. Es una cadena de caracteres de tamaño máximo permitido por la base de datos (*nvarchar(MAX)*). No permite valores *NULL*.
- **TipoEvento**: Define el tipo de evento que ha producido el mensaje. Es una cadena de caracteres de tamaño máximo 11 (*nvarchar(11)*). No permite valores *NULL*.

En la figura **(Figura 4-5)** se muestra el diseño de la tabla en el Administrador de SQL Server.

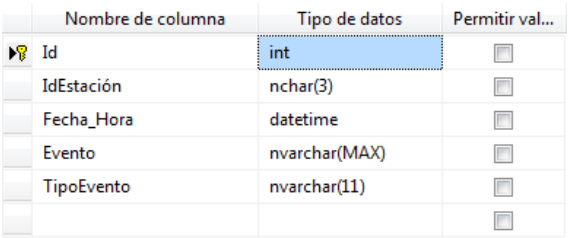

Figura 4-5. Tabla EVENTOS

4.6 ALARMAS

La tabla ALARMAS contiene un listado de alarmas que pueden producirse en las estaciones y que los PLC comunican al centro de control. Las alarmas producidas no tienen por qué ser las mismas en la red SAICA y en la red REACAR, pero el formato dado en las bases de datos es similar. Los campos que componen la tabla ALARMA son los siguientes:

- **Id**: Es el identificador de la alarma. Es un entero (*int*) autoincremental que funciona como llave principal de la tabla. No permite valores *NULL*.
- **Alarma**: Es el nombre de la alarma. Es una cadena de caracteres de tamaño máximo 30 (*nvarchar(30)*). No permite valores *NULL*.

En la figura **(Figura 4-6)** se muestra el diseño de la tabla en el Administrador de SQL Server.

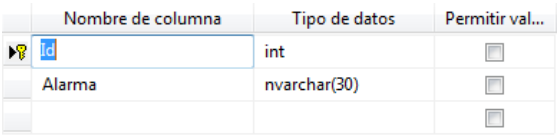

Figura 4-6. Tabla ALARMAS

4.7 A401 a A430 y A01 a A16

Las tablas de la A401 a la A430 en la base de datos SAICA y de la A01 a la A16 en la base de datos REACAR contienen el estado actual en el que se encuentran las alarmas de las estaciones. Todas las tablas tienen la misma estructura, tanto en la red SAICA como en la red REACAR, lo único que cambia es el nombre de las tablas. Los campos que contienen las tablas son los definidos a continuación.

- **Id**: Es el identificador de la alarma. Es un entero (*int*) que está relacionado con el campo Id de la tabla ALARMA. Es la llave principal de la tabla. No permite valores *NULL*.
- **Estado**: Contiene el estado de la alarma en la estación. Es un valor binario (*bit*) que indica que la alarma está activa cuando es *true* o inactiva cuando su valor es *false*. No permite valores *NULL*.

En la figura **(Figura 4-7)** se muestra el diseño de la tabla en el Administrador de SQL Server.

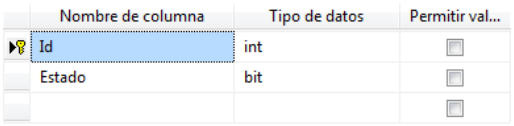

Figura 4-7. Tabla de alarmas A401 de la estación 401 de la base de datos SAICA

4.8 E401 a E430 y E01 a E16

Las tablas donde se almacenan los eventos producidos por las alarmas de las estaciones son las tablas E401 a E430 de la base de datos SAICA y las tablas E01 a E16 de REACAR. Estas tablas existen para mantener un registro de los cambios de estados de las alarmas de las estaciones.

- **Indice**: Es un índice que realiza la función de llave principal de la tabla. Es un entero (*int*) autoincremental. No permite valores *NULL*.
- **Fecha_Hora**: Contiene la hora y la fecha en la que ha sucedido el evento. Es una variable de tipo *datetime*. No permite valores *NULL*.
- **IdAlarma**: Es el identificador de la alarma que provoca el evento. El campo es de tipo entero (*int*). Está relacionado con el campo Id de la tabla ALARMAS. No permite valores *NULL*.
- **Estado**: Este campo indica a qué estado ha pasado la alarma. Es una cadena de caracteres de tamaño máximo 15 (*nvarchar(15)*). No permite valores *NULL*.

En la figura **(Figura 4-8)** se muestra el diseño de la tabla en el Administrador de SQL Server.

Nombre de columna	Tipo de datos	Permitir val...
Indice	int	
Fecha_Hora	datetime	
IdAlarma	int	
Estado	nvarchar(15)	

Figura 4-8. Tabla de eventos E401 de la estación 401 de la base de datos SAICA

4.9 Relaciones de las tablas

Las tablas de las bases de datos de la red SAICA, al igual que las de la red REACAR, están relacionadas entre ellas mediante llaves foráneas. En los apartados anteriores se exponen qué campos son los que están relacionados entre las distintas tablas. En este apartado se explicará las relaciones y lo que ello conlleva. Las relaciones de la base de datos REACAR son iguales a las relaciones de la base de datos SAICA, por lo que sólo se explican las relaciones de esta última.

En la figura **(Figura 4-9)** se muestra un esquema con las relaciones existentes entre las distintas tablas.

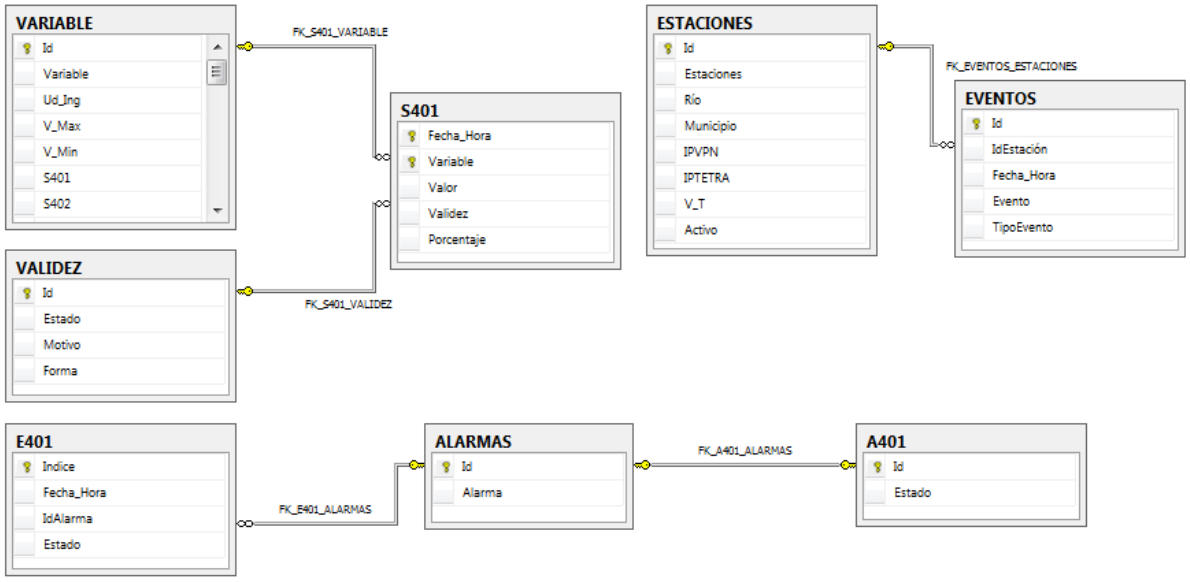

Figura 4-9. Relaciones en la base de datos SAICA

Las llaves foráneas son campos de otras tablas que se relacionan con campos de la propia tabla. Las relaciones conllevan varias restricciones en el manejo de las tablas. La primera restricción es que el campo al que ha sido asignada la llave foránea sólo puede tomar como valor los valores que contenga el campo que funciona como llave. Debido a esto, la segunda restricción es que SQL Server no permite eliminar una fila que contenga un valor que haya sido utilizado en el campo relacionado. Por ejemplo, el campo Validez de la tabla S401 puede contener únicamente los valores contenidos por el campo Id de la tabla VALIDEZ. Si un registro de la tabla VALIDEZ contiene un valor en el campo Id que esté contenido en el campo Validez de la tabla S401, éste registro no puede ser eliminado hasta que ese valor deje de existir en todas las tabla que estén relacionadas con ese campo. Este tipo de relación también implica que conociendo el valor almacenado en el campo relacionado de un registro, se puede conocer otros parámetros relacionados con ese valor. Por ejemplo, conociendo el valor en el campo Variable de un registro de la tabla S401, se conoce también de qué variable se trata, cuál es su unidad de ingeniería, etc.

Estas características afectan a todas las relaciones creadas en estas bases de datos. A continuación se explican las relaciones existentes.

Las tablas de estaciones (de S401 a S430) contienen dos llaves foráneas. La primera llave foránea es el campo Id de la tabla VARIABLES y está relacionada con el campo Variable. La segunda llave foránea es el campo Id de la tabla VALIDEZ. Está asignada al campo Validez.

La tabla EVENTOS tiene una llave foránea. Esta llave es el campo Id de la tabla ESTACIONES y está asignada al campo IdEstación.

En las tablas de eventos de estaciones (de E401 a E430) la llave foránea existente es el campo Id de la tabla ALARMAS. Esta llave relaciona los registros a través del campo IdAlarma. A su vez, el campo Id de la tabla ALARMAS también es llave foránea de las tablas A401 a A430. Esta relación es distinta a las anteriores puesto que la llave foránea está asociada a la llave principal de estas tablas, por lo que la relación en este caso es una a una, es decir, cada fila de la tabla ALARMA sólo puede estar relacionada a una fila de cada tabla de estación (de A401 a A430). En las demás relaciones, un valor de una llave foránea puede ser utilizado tantas veces se necesite en las tablas con las que está relacionada.

5 PROGRAMA DE VALIDACIÓN

Los programas de validación son las aplicaciones que gestionan el control de la red. Realizan la comunicación con las estaciones de medidas, gestionan la base de datos y son la herramienta que utiliza el técnico del centro de control para la revisión de los datos. Para explicar su funcionamiento, esta memoria divide el programa en tres partes: la comunicación con la base de datos, la comunicación con las estaciones y el funcionamiento de la interfaz gráfica. Como pasa con toda la memoria, el programa de validación de la red REACAR es una versión reducida del programa de validación de la red SAICA. Por este motivo, sólo se expondrá en este documento el funcionamiento del programa más complejo.

5.1 Comunicación con la base de datos

Visual Studio ofrece una funcionalidad que permite crear conexiones con bases de datos dentro de los proyectos en desarrollo. Esto se realiza a través de un asistente para la conexión de orígenes de datos. Al crear una conexión a una base de datos a través de esta funcionalidad, el desarrollador puede utilizar una serie de herramientas para realizar la comunicación de la aplicación con la base de datos de forma más sencilla.

Cuando se utiliza este método, Visual Studio crea una clase *DataSet* dentro del proyecto de desarrollo. Esta clase contiene la estructura de la base de datos y contiene distintos controles y funciones para su uso. La clase *DataSet* es una representación de datos residentes en memoria que proporciona un modelo de programación relacional coherente independientemente del origen de datos que contiene.

El desarrollador también puede crear conexiones con la base de datos a través de código. En este proyecto se utilizan ambos métodos para distintas funcionalidades. El primer método se utiliza para funciones sencillas y rápidas como lectura de datos. El segundo método se ha utilizado para la ejecución de transacciones puesto que en la depuración de la ejecución es más sencillo visualizar donde se producen errores y los motivos de ellos.

La clase creada es la clase *SAICA_DBDataSet*. Esta clase contiene las tablas y relaciones de la base de datos a la que está conectado el proyecto de Visual Studio. En la figura **(Figura 5-1)** se muestra la estructura de la clase. Como se puede apreciar es idéntica a la estructura de la base de datos SAICA.

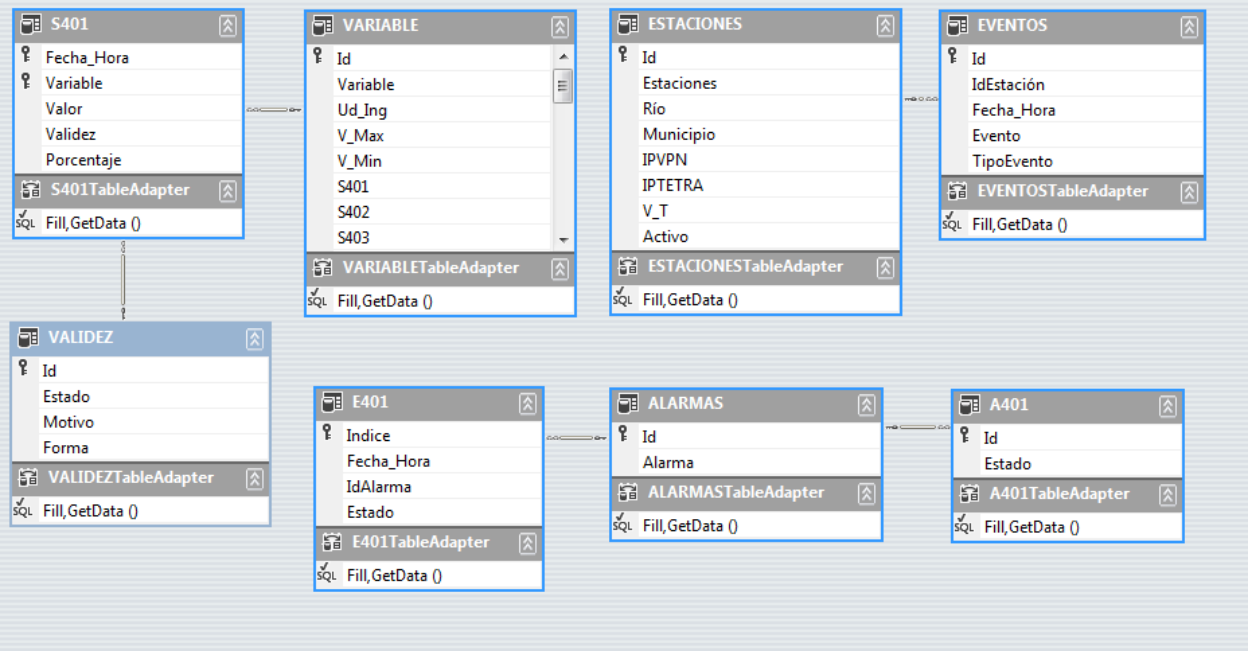

Figura 5-1. Estructura de la clase *SAICA_DBDataSet*

Una vez creada la clase *SAICA_DBDataSet*, se puede agregar en los formularios distintos controles *DataSet* de esa clase. La clase *DataSet* incluye la colección *DataTableCollection* de tabla de datos y la colección *DataRelation*. La clase *DataTable* incluye la colección *DataRowCollection* de filas de la tabla, la colección *DataColumnCollection* de columnas de datos y las colecciones *ChildRelation* y *ParentRelation* de relaciones de datos.

De forma predeterminada, un conjunto de datos no contiene datos reales. Rellenar un conjunto de datos significa cargar datos en los objetos *DataTable* individuales que constituyen el conjunto de datos, en este caso, *SAICA_DBDataSet*. Para rellenar un conjunto de datos se utiliza ejecutando consultas de *TableAdapter*. Particularmente, para rellenar los distintos *DataTable* se utiliza el método *Fill* de los *TableAdapter* tal y como se muestra en la figura **(Figura 5-2)**.

```
this.a401TableAdapter.Fill(this.sAICA_DBDataSet.A401);
this.s401TableAdapter.Fill(this.sAICA_DBDataSet.S401);
this.eSTACIONESTableAdapter.Fill(this.sAICA_DBDataSet.ESTACIONES);
this.vALIDEZTableAdapter.Fill(this.sAICA_DBDataSet.VALIDEZ);
this.vARIABLESTableAdapter.Fill(this.sAICA_DBDataSet.VARIABLES);
this.eVENTOSTableAdapter.Fill(this.sAICA_DBDataSet.EVENTOS);
this.e401TableAdapter.Fill(this.sAICA_DBDataSet.E401);
this.aLARMASTableAdapter.Fill(this.sAICA_DBDataSet.ALARMAS);
```

Figura 5-2. Método *Fill* de *TableAdapter*

Una vez rellenas las tablas de s*AICA_DBDataSet*, se pueden realizar diferentes acciones mediante los métodos de los que dispone la clase. En la siguiente figura **(Figura 5-3)** se muestra como se realiza la consulta del campo Id y del campo Estaciones de la tabla ESTACIONES. *IdEstacion* y *Estacion* son dos variables tipo *string*.

```
IdEstacion = sAICA_DBDataSet.ESTACIONES[selIndexEstacion].Id;
Estacion = sAICA_DBDataSet.ESTACIONES[selIndexEstacion].Estaciones;
```

Figura 5-3. Lectura de la base de datos

Para crear un nuevo registro existe el método *DataTable.NewRow*. Este método genera una nueva fila del *DateTable* que debe ser rellenado antes de añadirla mediante el método *DataTable.Row.Add()*. En la figura **(Figura 5-4)** se muestra como se realiza esta acción.

```
// Crear objeto fila
DataRow Fila = this.sAICA_DBDataSet.EVENTOS.NewEVENTOSRow();
// Rellenar valores de columnas de la nueva fila
Fila["IdEstacion"] = IdER;
Fila["Fecha_Hora"] = DateTime.Now;
Fila["Evento"] = MensajeEv;
Fila["TipoEvento"] = TipoEv;
// Añadir fila
this.sAICA_DBDataSet.Eventos.Rows.Add(Fila);
```

Figura 5-4. Métodos *DateTable.NewRow* y *DateTable.Row.Add()*

La modificación de un campo dentro de un registro de una tabla se realiza utilizando las sentencias mostradas en la figura **(Figura 5-5)**. Primero se asocia la fila que se pretende modificar a una fila. A continuación, se introduce un valor en el campo a modificar.

```
DataRow row = sAICA_DBDataSet.Tables["S" + IdEstacion].Rows.Find(DateTime.FromOADate(x), IdVariable);
row["Validez"] = ParamValida;
```

Figura 5-5. Edición de un campo de un registro

Para eliminar una fila, se utiliza el método *DateTable.Rows[].Delete* tal como se muestra en el ejemplo siguiente **(Figura 5-6)**:

```
sAICA_DBDataSet.Eventos.Rows[i].Delete();
```

Figura 5-6. Borrado de una fila

Todos los métodos mostrados (añadir fila, editarla o eliminarla) no tienen efecto en la base de datos. Para almacenar todos los cambios en la base de datos se debe llamar al método *Update* de *TableAdapter*. De esta forma, el programa realiza una consulta a la base de datos de forma que aplique los cambios realizados a ésta. En la siguiente figura **(Figura 5-7)** se muestra cómo se escribe la sentencia:

```
this.eventosTableAdapter.Update(this.sAICA_DBDataSet.Eventos);


this.s401TableAdapter.Update(row);
```

Figura 5-7. Método *TableAdapter.Update()*

Al método *Update* se le puede introducir como argumento una única fila, todo el *DateTable* que se quiera actualizar o el *DataSet* completo.

```
string sConexion = "SERVER=SAICA-SERVER;DATABASE=SAICA_DB;Trusted_Connection=true";
SqlConnection sqlConexion = new SqlConnection(sConexion);

SqlDataAdapter sqlDAS401 = new SqlDataAdapter();

//Creación de comando de lectura para la tabla S401
SqlCommand command = new SqlCommand("SELECT * FROM S401", sqlConexion);
sqlDAS401.SelectCommand = command;

//Creación de comando de creación de fila para la tabla s401
command = new SqlCommand("INSERT INTO S401 (Fecha_Hora, Variable, Valor, Validez, Porcentaje)" +
    "VALUES (@Fecha_Hora, @Variable, @Valor, @Validez, @Porcentaje)", sqlConexion);
command.Parameters.Add("@Fecha_Hora", SqlDbType.DateTime);
command.Parameters.Add("@Variable", SqlDbType.Int);
command.Parameters.Add("@Valor", SqlDbType.Real);
command.Parameters.Add("@Validez", SqlDbType.Int);
command.Parameters.Add("@Porcentaje", SqlDbType.Real);
sqlDAS401.InsertCommand = command;

//Creación de comando de edición de fila para la tabla s401
command = new SqlCommand("UPDATE S401 SET Validez = @Validez " +
    "WHERE Fecha_Hora = @Fecha_Hora AND Variable = @Variable", sqlConexion);
command.Parameters.Add("@Fecha_Hora", SqlDbType.DateTime);
command.Parameters.Add("@Variable", SqlDbType.Int);
command.Parameters.Add("@Validez", SqlDbType.Int);
sqlDAS401.UpdateCommand = command;
```

Figura 5-8. Creación de *TableAdapter*

Para poder crear conexiones con una base de datos SQL de forma manual se debe añadir la librería *System.Data.SqlClient*. Para acceder a los datos de la base de datos, primero se ha de crear una cadena de conexión con el servidor SQL y la base de datos. Una vez se tiene la conexión con la base de datos, hay que definir los métodos de un *TableAdapter* para poder realizar las consultas con la base de datos. En la figura

(Figura 5-8) se muestra como se crea un *TableAdapter* con los comandos *Select*, *Insert* e *Update* para la comunicación con la tabla S401.

Una vez se han creado los métodos, para extraer los datos de la base de datos se utiliza el método *Fill()* y el método *Update()* para la modificación de la base de datos, tal y como se utiliza en el *TableAdapter* generado por el asistente para la conexión de orígenes de datos ya explicado. En la figura **(Figura 5-9)** se muestra el ejemplo del método *Fill()*.

```
//Extracción de los datos de la estación 401
DataTable dtS401 = new DataTable();
sqlDAS401.Fill(dtS401);
```

Figura 5-9. Método *TableAdapter.Fill()* para comunicación con base de datos manual

También se puede realizar consultas *INSERT* o *UPDATE* a la base de datos a través del método *SqlCommand.ExecuteNonQuery*. Para ello, se crea el objeto *SqlCommand* tal y como se ve en la figura anterior **(Figura 5-8)**. Una vez creada sólo hay que llamar al método y la consulta se ejecuta. En la siguiente figura **(Figura 5-10)** se muestra como se realiza una consulta *UPDATE*.

```
//Crear la consulta
sComando = "UPDATE A" + IdER + " SET Estado = @Estado" +
    " WHERE Id = @Id";
sqlComando.CommandText = sComando;
sqlComando.Parameters.AddWithValue("@Estado", Datos.PaxBit[i]);     //Estado
sqlComando.Parameters.AddWithValue("@Id", i + 1);                   //IdAlarma
//Ejecutar la consulta
sqlComando.ExecuteNonQuery();
```

Figura 5-10. Consulta *UPDATE*

Para la ejecución de la consulta *SELECT*, se utiliza el método *SqlCommand.ExecuteReader*. Este método devuelve un objeto *SqlReader* que contiene las filas solicitadas por la consulta. En este proyecto se utiliza este método filtrando la consulta por datos inequívocos, por lo que el objeto *SqlReader* devuelto sólo contendrá una única fila. En la figura **(Figura 5-11)** se muestra como se realiza este proceso.

```
//Crear la consulta
sComando = "SELECT * FROM A" + IdER +" WHERE Id = @Id";
sqlComando.CommandText = sComando;
sqlComando.Parameters.AddWithValue("@Id", i + 1);                   //IdAlarma
//Ejecutar la consulta
sqlUltimoEstado = sqlComando.ExecuteReader();
```

Figura 5-11. Consulta *SELECT*

Cuando se requiere utilizar varias consultas a la base de datos de forma que estas se realicen como una única consulta, SQL Server permite utilizar las transacciones. Para la ejecución de una transacción en C#, el primer paso a dar es la creación de una conexión con la base de datos. A partir de esta conexión se inicia la transacción con el método *SqlConnection.BeginTransaction* creando un objeto *SqlTransaction* que será el que contenga la información de la transacción. Una vez creada la transacción, se crea un objeto *SqlCommand* para enviar las consultas. A este objeto se le asocia la transacción y la conexión con la base de datos. Después de terminar la configuración de la transacción, se realizan las consultas dentro de un bloque *try-catch* para que se capture cualquier excepción que pueda suceder durante la comunicación. Toda la configuración de la transacción es mostrada a continuación **(Figura 5-12)**.

```
using (SqlConnection sqlConexion = new SqlConnection(sConexion))
{
    sqlConexion.Open();
    SqlCommand sqlComando = sqlConexion.CreateCommand();
    SqlTransaction sqlTrans;

    //Comienzo de transacción
    sqlTrans = sqlConexion.BeginTransaction();
    sqlComando.Connection = sqlConexion;
    sqlComando.Transaction = sqlTrans;

    try
    {
```

Figura 5-12. Configuración de una transacción

Cuando se han realizado todas las consultas de la transacción, se finaliza ésta con el método *SqlTransaction.Commit*. Este método es llamado aún dentro del bloque *try*. Si se ha producido alguna excepción durante la transacción, se ejecuta el método *SqlTransaction.Rollback* que deshace todos los cambios

que se han efectuado en la base de datos dentro de la transacción. La figura **(Figura 5-13)** muestra la finalización de la transacción.

```
            //Se realiza la transacción
            sqlTrans.Commit();
        }
        catch (Exception ex)
        {
            //Realizar RollBack para deshacer lo anterior al error
            try
            {
                sqlTrans.Rollback();
            }
            catch (Exception ex2)
            {
                //La ejecución del método Rollback ha fallado
            }
        }
    }
```

Figura 5-13. Finalización de una transacción

5.2 Comunicación con la red de estaciones

Para la comunicación con las estaciones de medida se utilizan las herramientas que proporciona OMRON con el software CX-Compolet. Para ello, se ha de tener instalado dicho software en Visual Studio. Una vez instalado, se hace uso de estas herramientas añadiendo las librerías necesarias dentro del código en tiempo de desarrollo. En la figura **(Figura 5-14)** se muestra como se añade la librería que se ha usado en este proyecto.

```
using OMRON.Compolet.CIP;
```

Figura 5-14. Librería de comunicación con dispositivos OMRON

Como se ha explicado en el apartado **3.1 Diseño del sistema**, por cada estación con la que se va a comunicar se crea un hilo independiente que se ejecuta paralelamente al hilo principal (hilo padre), y a los otros hilos de comunicación. Para poder utilizar subprocesos, es necesario incluir las librerías de sistemas que contienen estas herramientas. En la figura **(Figura 5-15)** se muestran las librerías agregadas.

```
using System.Threading.Tasks;
using System.Threading;
```

Figura 5-15. Librerías para multihilo

La lectura de datos se produce por un control *Timer* que ha sido configurado para que genere un evento cada minuto. Dentro de este evento, se comprueba si es el momento de realizar la lectura. La lectura de los datos de las estaciones se producen cada 15 minutos comenzando a contar desde las 00:03. La función que ejecuta el evento del *Timer* es la que muestra la figura **(Figura 5-16)**.

```
// COMUNICACIÓN CON PLC Y GUARDAR DATOS EN BD: Comunicar a la hora determinada
private void tiCommPLC_Tick(object sender, EventArgs e)
{
    dtActual = DateTime.Now;
        // Leer datos cada x minutos (periodo) empezado a horas en punto con un offset
        // Offset para dar tiempo a escribir los datos al PLC
        // PLC escribe los datos con el mismo periodo empezando a horas en punto sin offset
        // De momento periodo y offset fijos en 15 y 3 minutos respectivamente
        if (((dtActual.Minute - 3) % 15) == 0)//((((dtActual.Minute-offset)%periodo)==0)
        {
            HiloAsyncMed();
        }

}
```

Figura 5-16. Evento *Timer_Tick*

La función *HiloAsyncMed()* es la función que inicia los hilos de procesos de lectura de datos. En esta función se crean subprocesos para las estaciones activas. Los nuevos hilos comienzan ejecutando la función *Lectura* a la que se le introduce como parámetro el número de estación remota (*ER*) a través del método *Thread.Start()*. En la figura **(Figura 5-17)** se muestra la función *HiloAsyncMed()*.

```
public void HiloAsyncMed()
{
    int ER = 0;
    string IdER;

    this.eSTACIONESTableAdapter.Fill(this.sAICA_DBDataSet.ESTACIONES);

    // Evaluar cuales son las estaciones operarivas
    // Activar las conexiones y si no da error arrancar subproceso (hilo)
    for (ER = 0; ER < this.sAICA_DBDataSet.ESTACIONES.Rows.Count; ER++)
    {
        IdER = SaicaId(ER); //Devuelve la Id de la estación
        if (this.sAICA_DBDataSet.ESTACIONES.FindById(IdER).Activo)
        {
            Thread thLectura = new Thread(Lectura);
            thLectura.IsBackground = true;
            thLectura.Start(ER);
        }
    }
}
```

Figura 5-17. Función *HiloAsyncMed*

En la función *Lectura* se inicia la herramienta de conexión con los PLC (clase *CJ2Compolet*). Con los métodos *try-catch* se realiza la monitorización de errores en tiempo de ejecución. Este método ejecuta las sentencias que se escriben dentro de la parte *try*. Si se produce alguna excepción durante la ejecución de la aplicación, se detiene la ejecución de código dentro del módulo *try* y se ejecuta el módulo *catch* que coincida con el tipo de excepción que se haya producido. La clase *CancellationTokenSource* inicia un temporizador para que la ejecución de la comunicación se detenga tras un tiempo, en este caso, 5 minutos. Una vez se inicia el temporizador, se llama a la función *ObtenerDatosAsync* y se espera a que acabe. Cuando se produce alguna excepción, se escribe en la tabla EVENTOS de la base de datos un registro con el error que la ha producido. En la siguiente figura **(Figura 5-18)** se muestra el código de la función *Lectura*.

```
public async void Lectura(int ER)
{
    CancellationTokenSource ERcts = new CancellationTokenSource();
    CJ2Compolet compoletS = ini_compolet();
    // Proteger tarea asíncrono
    try
    {
        // Cancelar tarea asíncrona por tiempo (5 minutos)
        ERcts.CancelAfter(300000);
        // Esperar a que obtenga los datos de una estación
        await ObtenerDatosAsync(ER, ERcts.Token, compoletS);
    }
    catch (OperationCanceledException ex)
    {
        // Guardar evento
        AddEvento(ER, ex.Message, "ERROR");
    }
    catch (TransactionAbortedException ex)
    {
        // Guardar evento
        AddEvento(ER, ex.Message, "ERROR");
    }
    catch (Exception ex)
    {
        // Guardar evento
        AddEvento(ER, ex.Message, "ERROR");
    }
    // Desactivar conexión
    if (compoletS.Active == true)
    {
        compoletS.Active = false;
    }
}
```

Figura 5-18 Función *Lectura*

La función *ObtenerDatosAsync* convierte en asíncrona la función síncrona *ObtenerDatos*. Esto lo realiza mediante el método *Task.Run()*. El código es el mostrado en la figura **(Figura 5-19)**.

```
// Tarea asíncrona (subproceso en segundo plano): Obtener datos de Estación Remota i
private async Task ObtenerDatosAsync(int ER, CancellationToken ct, CJ2Compolet compoletS)
{
    // Convertir tarea síncrona en asíncrona y esperar ejecución
    await Task.Run(() => ObtenerDatos(ER, ct, compoletS));
}
```

Figura 5-19. Función *ObtenerDatosAsync*

Para extraer los datos del PLC es necesario conocer cómo están almacenados.

La memoria del PLC está dedicada a dos propósitos. El primero es a almacenar el programa que ejecuta el PLC, el cual no es necesario conocer para el objetivo de este proyecto. El segundo propósito es almacenar los datos obtenidos de la red de sensores. Para este fin se reserva una parte de la memoria donde se ordenan los datos de forma estructurada. En la figura **(Figura 5-20)** se muestra la ordenación de la memoria del PLC.

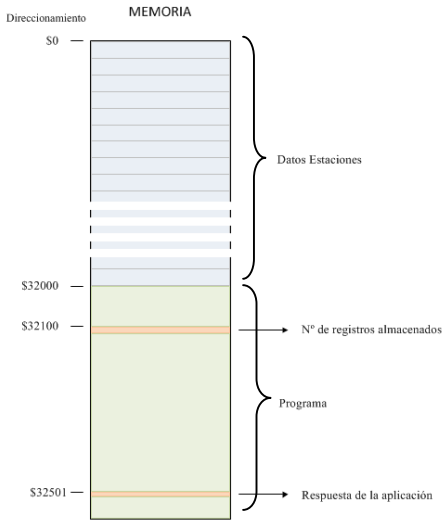

Figura 5-20. Estructura de la memoria del PLC

El PLC realiza una lectura de datos de la red de sensores cada 15 minutos y los almacena en la memoria. Los datos leídos son ordenados en paquetes de datos con la estructura representada en la siguiente tabla **(Tabla 3)**:

Tabla 3. Paquete de registros del PLC de una estación SAICA

Año
Mes
Día
Hora
Minuto
Alarmas
pH
Temperatura Agua
Turbidez
Oxígeno disuelto
Conductividad
Redox
Amonio
Nitrato
Fosfato
Potasio
Cloruro
Carbono orgánico
Temperatura Caseta
Nivel Río
Caudal Río
Reserva 1
Reserva 2

Los paquetes de datos son almacenados en memoria de forma correlativa, comenzando desde la dirección *$0*. El tamaño de cada paquete es de 40 registros. Las variables de fecha (Año, Mes, Día, Hora, Minuto) ocupan 1 registro cada una. Los estados de las alarmas están contenidos en un único registro. Los valores de los datos de medida ocupan cada uno 2 registros. Cada vez que se añade un paquete, el PLC modifica un registro interno que contiene el número de registros almacenados. Cuando el espacio de memoria reservado para el almacenamiento de datos está completo, el PLC comienza a almacenar los paquetes desde la dirección *$0*, sobrescribiendo los datos más antiguos. En el momento que la aplicación de validación finaliza la lectura de datos del PLC satisfactoriamente, la aplicación modifica un registro en el PLC poniéndolo a *1*. Cuando esto ocurre, el PLC pone el registro que contiene el número de registros y el registro modificado por la aplicación a *0*. De esta forma, el PLC comienza a almacenar los nuevos paquetes desde la dirección *$0*.

Tras entender el funcionamiento del PLC, la función *ObtenerDatos* debe conectarse al PLC, leer del PLC el número de registros que debe extraer, extraer los registros del PLC, almacenar los datos extraídos en la base de datos y escribir en el registro de respuesta en el PLC un *1*.

Para realizar la conexión con un PLC, la función *ObtenerDatos* realiza una llamada a la función *ActivarComm*. Esta función realiza tres intentos de conexión con el PLC a través de la dirección IP predeterminada (TETRA o VPN), tal y como se muestra en la figura **(Figura 5-21)**.

```
//Activar conexión con Estación Remota
private CJ2Compolet ActivarComm(int ER, CJ2Compolet compoletS)
{
    int i;
    string IdER;
    bool Conectado;
    string TipoPLC;
    string NombrePLC;

    IdER = SaicaId(ER);

    // Por las dos vías de comunicación.
    // Si falla la activa, se notifica, pero se intenta por la siguiente
    // ERROR sólo si falla por las dos

    // Primero 3 intentos
    for (i = 0; i < 3; i++)
    {
        if (compoletS.Active == false)
        {
            try
            {
                //Toma la dirección IP del método preferente
                if(this.sAICA_DBDataSet.ESTACIONES.FindById(IdER).V_T)
                    compoletS.PeerAddress = sAICA_DBDataSet.ESTACIONES.FindById(IdER).IPTETRA;
                else
                    compoletS.PeerAddress = sAICA_DBDataSet.ESTACIONES.FindById(IdER).IPVPN;
                compoletS.Active = true;
                //Acciones para comprobar que la conexión se ha realizado
                Conectado = compoletS.IsConnected;
                TipoPLC = compoletS.TypeName;
                NombrePLC = compoletS.UnitName;
            }
            catch (Exception)
            {
                compoletS.Active = false;
                // Si se han agotado los 3 intentos, guarda evento: Fallo
                if (i == 2)
                {
                    if (this.sAICA_DBDataSet.ESTACIONES.FindById(IdER).V_T)
                        AddEvento(ER, "Fallo conexión TETRA", "Advertencia");
                    else
                        AddEvento(ER, "Fallo conexión VPN", "Advertencia");
                }
            }
        }
    }
```

Figura 5-21. Función *AcctivarComm*, primera parte

Si los tres intentos fallan, se escribe el fallo de conexión en la tabla EVENTOS de la base de datos y se realizan otros tres intentos de conexión a través de la otra dirección IP almacenada. Si los segundos tres intentos no permiten la conexión, se suspende el proceso de lectura y se introduce en la tabla EVENTOS de la base de datos el error de conexión. Si en alguno de los intentos se ha podido establecer la conexión, la función devuelve el objeto CJ2Compolet configurado a la función *ObtenerDatos*. En la figura **(Figura 5-22)** se muestra el código de la segunda parte de la función *ActivarComm*.

```
    // Si fallan los 3 primeros intentos, intentar otro método
    // 3 reintentos. Si falla el tercero se captura el error con la llamada asíncrona
    if (compoletS.Active == false)
    {
        for (i = 0; i < 2; i++)
        {
            if (compoletS.Active == false)
            {
                try
                {
                    //Se toma la dirección IP del método NO preferente
                    if (this.sAICA_DBDataSet.ESTACIONES.FindById(IdER).V_T)
                        compoletS.PeerAddress = sAICA_DBDataSet.ESTACIONES.FindById(IdER).IPVPN;
                    else
                        compoletS.PeerAddress = sAICA_DBDataSet.ESTACIONES.FindById(IdER).IPTETRA;
                    compoletS.Active = true;
                    //Acciones para comprobar que la conexión se ha realizado
                    Conectado = compoletS.IsConnected;
                    TipoPLC = compoletS.TypeName;
                    NombrePLC = compoletS.UnitName;
                }
                catch (Exception)
                {
                    compoletS.Active = false;

                }
            }
        }
    }
    // reintento 3
    // No se utiliza bloque try/catch para capturar el error con el de la llamada asíncrona a este méto
    if (compoletS.Active == false)
    {
        //Se toma la dirección IP del método NO preferente
        if (this.sAICA_DBDataSet.ESTACIONES.FindById(IdER).V_T)
            compoletS.PeerAddress = sAICA_DBDataSet.ESTACIONES.FindById(IdER).IPVPN;
        else
            compoletS.PeerAddress = sAICA_DBDataSet.ESTACIONES.FindById(IdER).IPTETRA;
        compoletS.Active = true;
        //Acciones para comprobar que la conexión se ha realizado
        Conectado = compoletS.IsConnected;
        TipoPLC = compoletS.TypeName;
        NombrePLC = compoletS.UnitName;
    }

    return compoletS;
}
```

Figura 5-22. Función *ActivarComm*, segunda parte

La función *ini_compolet* que aparece en la figura **(Figura 5-18)**, inicializa un objeto *CJ2Compolet* y lo devuelve. La conexión se realiza por UCMM. UCMM es el protocolo que utiliza SYSMAC Gateway para conectarse con los PLC de OMRON. Se configura un tiempo límite de recepción de 5 segundos. El puerto local seleccionado es el puerto que se ha configurado en SYSMAC Gateway como puerto de comunicaciones. La configuración de SYSMAC Gateway se explica en el **Anexo III**. A continuación se muestra la función *ini_compolet* en la figura **(Figura 5-23)**.

```
// INICIALIZACIÓN DE COMPONENTES COMPOLET.
public void ini_compolet()
{
    CJ2Compolet compoletS = new CJ2Compolet();
    compoletS.Active = false;
    compoletS.ConnectionType = ConnectionType.UCMM;
    compoletS.HeartBeatTimer = 0;
    compoletS.LocalPort = 2;
    compoletS.ReceiveTimeLimit = 5000;
    compoletS.UseRoutePath = false;

    return compoletS;
}
```

Figura 5-23. Función *ini_compolet*

Una vez se ha establecido la conexión con el PLC, la función *ObtenerDatos* lee el registro de la dirección *$32100* de la memoria del PLC donde se almacena el número de registros que debe leer la aplicación en esta conexión. La lectura de dicho registro se realiza a través del método que se muestra en la figura **(Figura 5-24)**. El método *ReadMemoryDwordLong* del objeto *CJ2Compolet* necesita como argumentos el tipo de memoria a la que accede, la dirección de memoria donde comenzar a leer. Opcionalmente, se puede incluir como argumentos el número de registros a leer y el tipo de dato que se lee. Una vez realiza la lectura, devuelve un

array de enteros. Cuando se omite el número de registros a leer, por defecto sólo lee uno y el *array* de enteros es de tamaño 1.

```
//Leer número de registros a leer
NumReg = (int)compoletS.ReadMemoryDwordLong(CJ2Compolet.MemoryTypes.ExDM0, 32100, OMRON.Compolet.CIP.DataTypes.SBIN);
```

Figura 5-24. Lectura de registro a través del método *CJ2Compolet.ReadMemoryDwordLong()*

Tras la lectura del número de registros a leer, se calcula el número de paquetes contenidos en el PLC. El tamaño de un paquete de datos es igual al de 40 registros. Si el número de registros obtenido es mayor a 32000, quiere decir que se ha superado el tamaño de la memoria reservada para datos y se han sobrescrito algunos. Esto conlleva que el número de paquetes a extraer es 800 y que la dirección de memoria del PLC por la que hay que empezar a leer ya no es la dirección *$0* y se debe calcular. En la figura **(Figura 5-25)** se muestra este proceso.

```
// Calcular dirección inicial
if (NumReg > 32000)
{
    NumPax = 800; // 32000 / 40
    DirIni = NumReg - 32000;
}
else
{
    // Calcular número de paquetes
    NumPax = NumReg / 40;
    DirIni = 0;
}
```

Figura 5-25. Calculo de número de paquetes y de la dirección de inicio de lectura

Para la lectura de los paquetes de datos se ha creado un *struct* con el que trabajar con los datos extraídos de la estación de medida. Está compuesto por una variable *datetime*, un *array* de variables tipo *bool* y un *array* de variables tipo *float*, tal y como se muestra en la figura **(Figura 5-26)**.

```
public struct PaxStruct
{
    public DateTime PaxFecha;
    public bool[] PaxBit;
    public float[] PaxMed;
}
```

Figura 5-26. Estructura *PaxStruct*

Después de calcular el número de paquetes a leer y la dirección de memoria por donde comenzar, se realiza un bucle donde se va extrayendo los datos del PLC. Este bucle es el mostrado en la figura **(Figura 5-27)**. El bucle comienza a leer en la dirección de inicio. Realiza tantas lecturas como paquetes de datos haya. Cuando se llega al final de la memoria, sigue leyendo desde la dirección *$0*. La lectura la realiza la función *LeerPax*. Esta función devuelve el paquete de datos que se encuentra en la dirección que recibe como argumento (*oofPax*).

```
// Crear Array dinámico de NumPax elementos de estructura de paquete de datos leido
//Se libera automáticamente cuando acaba el proceso
PaxEstru = new PaxStruct[NumPax];

// Obtener paquetes de datos
offPax = DirIni;
for (i = 0; i < NumPax; i++)
{
    PaxEstru[i] = LeerPax(offPax, compoletS);
    offPax += 40;
    if (offPax >= 32000)
        offPax = 0;
}
```

Figura 5-27. Bucle de lectura de datos

La función *LeerPax* lee primero la fecha contenida en el paquete de datos y almacena la fecha en la variable *PaxFecha* de la estructura. A continuación, realiza la lectura de los estados de las alarmas para almacenarlos en el *array PaxBit* de la estructura. Por último, extrae los valores de las medidas y los guarda en el *array PaxMed*. La función es mostrada en la figura **(Figura 5-28)**.

```
//Leer paquete de datos de PLC
private PaxStruct LeerPax(int offPax, CJ2Compolet compoletS)
{
    PaxStruct DatoLeido;
    int[] DatoFecha;
    int offSet;
    offSet = offPax;
    //Lectura de la fecha de lectura de variables
    //Año, Mes, Día, Hora, Minuto
    DatoFecha = compoletS.ReadMemoryWordInteger(CJ2Compolet.MemoryTypes.ExDM0, offSet, 5, OMRON.Compolet.CIP.DataTypes.SBIN);
    DatoLeido.PaxFecha = new DateTime(DatoFecha[0], DatoFecha[1], DatoFecha[2], DatoFecha[3], DatoFecha[4], 0);
    offSet += 5; //Desplazamiento de 5 registros
    //Lectura de estados de las alarmas de la estación
    //16 bits
    DatoLeido.PaxBit = compoletS.ReadMemoryBit(CJ2Compolet.MemoryTypes.ExDM0, offSet, 0, 16);
    offSet += 1; //Desplazamiento de 1 registro
    //Lectura de los valores de las variables
    //17 variables
    DatoLeido.PaxMed = compoletS.ReadMemoryDwordSingle(CJ2Compolet.MemoryTypes.ExDM0, offSet, 17);
    return DatoLeido;
}
```

Figura 5-28. Función *PaxStruct*

Seguidamente, tras la obtención de los datos del PLC, la función *ObtenerDatos* llama a la función *GuardarDatos*. Finalmente, si no se ha producido ningún error en todo el proceso que lo haya finalizado antes de tiempo, escribe en el registro de respuesta del PLC como se muestra en la figura **(Figura 5-29)** para confirmar a este que la lectura se ha realizado satisfactoriamente.

```
// Escribir en PLC que la obtención de datos se ha realizado correctamente
this.compoletS[ER].WriteMemoryWordInteger(CJ2Compolet.MemoryTypes.ExDM0, 32501, 1, OMRON.Compolet.CIP.DataTypes.BIN);
```

Figura 5-29. Confirmación de lectura del PLC

La función *GuardarDatos* realiza la introducción de los nuevos datos en la base de datos. Para que todo el proceso de extracción de datos del PLC y escritura en la base de datos se realice como un proceso unitario, es decir, que el proceso se ejecute completo o que no se ejecute nada, la escritura en la base de datos se realiza a través de una transacción. Tal como muestra la figura **(Figura 5-30)**, la función crea una conexión con la base de datos y abre una transacción. Dentro de ésta, el primer paso es comprobar que los datos recibidos no se encuentren ya en la base de datos. La comprobación consiste en revisar que las fechas de los datos recibidos sean posteriores a la última fecha introducida en la tabla de la estación. Una vez se verifica que los datos a introducir son correctos, se realiza la introducción del paquete verificado a través de la función *EscribirMedidas*. Si el paquete de datos contiene una fecha ya incluida en la base de datos, la aplicación añade un registro en la tabla EVENTOS notificándolo. Después de introducir todos los datos de medidas de las variables, el siguiente paso es introducir los datos referentes a las alarmas de la estación. Esto lo realiza la función *EscribirAlarmas*. Cuando se ha terminado de escribir todos los datos capturados del PLC en la base de datos, se envía la transacción y se da ésta por finalizada. En el caso de suceder un error durante la transacción, la aplicación captura una excepción y ejecuta el método *SqlTransaction.Rollback* para deshacer lo ejecutado dentro de la transacción.

```
//Guardar datos leidos en la Base de datos
private void GuardarDatos(int ER, PaxStruct[] Datos)
{
    int i;
    string IdER = SaicaId(ER);
    string sConexion = "SERVER=SAICA-SERVER;DATABASE=SAICA_DB;Trusted_Connection=true";
    string sError;
    int NumPax = Datos.Length;
    DataRow ultimaFila;
    DateTime ultimaDT;

    using (SqlConnection sqlConexion = new SqlConnection(sConexion))
    {
        sqlConexion.Open();
        SqlCommand sqlComando = sqlConexion.CreateCommand();
        SqlTransaction sqlTrans;

        //Comienzo de transacción
        sqlTrans = sqlConexion.BeginTransaction();
        sqlComando.Connection = sqlConexion;
        sqlComando.Transaction = sqlTrans;

        try
        {
            //Insertar en la base de datos los datos de los paquetes leidos
            for (i = 0; i < NumPax; i++)
            {
                // Comprobar fecha
                if (this.sAICA_DBDataSet.Tables["S" + IdER].Rows.Count > 0)
                {
                    ultimaFila = this.sAICA_DBDataSet.Tables["S" + IdER].Rows[this.sAICA_DBDataSet.Tables["S" + IdER].Rows.Count - 1];
                    ultimaDT = Convert.ToDateTime(ultimaFila["Fecha_Hora"]); //Fecha de la ,ultima lectura almacenada
                    //Si la fecha de lectura es posterior a la última lectura almacenada,
                    //almacena el paquete
                    if (Datos[i].PaxFecha > ultimaDT)
                    {
                        EscribirMedidas(ER, Datos[i], sqlComando);
                    }
                    else //Si la fecha es anterior, no se almacena el paquete y se notifica en la tabla EVENTOS
                    {
                        AddEvento(ER, "Datos con la misma fecha", "Advertencia");
                    }
                }

            }
            //Insertar en la base de datos los nuevos eventos de estación y actualizar las alarmas
            for (i = 0; i < NumPax; i++)
            {
                EscribirAlarmas(ER, Datos[i], sqlComando);
            }

            //Se realiza la transacción
            sqlTrans.Commit();
        }
        catch(Exception ex)
        {
            //Realizar RollBack para deshacer lo anterior al error
            try
            {
                sqlTrans.Rollback();
            }
            catch (Exception ex2)
            {
                //Lanzar error para que lo capture la función Lectura()
                sError = "Error al almacenar los datos en la base de datos";
                sError += Environment.NewLine;
                sError += "Posterior error al cancelar la escritura de datos en la base de datos";
                throw new Exception(sError);
            }
            //Lanzar error para que lo capture la función Lectura()
            sError = "Error al almacenar los datos en la base de datos";
            throw new Exception(sError);
        }
    }
}
```

Figura 5-30. Función *GuardarDatos*

La función *EscribirMedidas* **(Figura 5-31)** utiliza la conexión abierta en la función *GuardarDatos* para insertar a través de consultas SQL nuevas filas en la tabla de la estación. Esta función es ejecutada dentro de una transacción, por lo que los cambios que realiza en la base de datos pueden ser anulados si sucede algún error que interrumpa la transacción.

```
//Escribir Tabla de Estaciones
private void EscribirMedidas(int ER, PaxStruct Datos, SqlCommand sqlComando)
{
    int i;
    int NumDatos;
    string IdER = SaicaId(ER);
    float fVMax, fVMin, fPorcentaje;
    string sComando;
    //Insertar filas con los valores de medida del paquete en la tabla de estación
    NumDatos = Datos.PaxMed.Length;
    for (i = 0; i < NumDatos; i++)
    {

        fVMax = this.sAICA_DBDataSet.VARIABLES.FindById(i + 1).V_Max; //Valor máximo de la variable
        fVMin = this.sAICA_DBDataSet.VARIABLES.FindById(i + 1).V_Min; //Valor mínimo de la variable
        fPorcentaje = ((Datos.PaxMed[i] - fVMin) * 100) / (fVMax - fVMin);

        //Crear la consulta
        sComando = "INSERT INTO S" + IdER + " (Fecha_Hora, Variable, Valor, Validez, Porcentaje)" +
            " VALUES (@Fecha_Hora, @Variable, @Valor, @Validez, @Porcentaje)";
        sqlComando.CommandText = sComando;
        sqlComando.Parameters.AddWithValue("@Fecha_Hora", Datos.PaxFecha);  //Fecha
        sqlComando.Parameters.AddWithValue("@Variable", i + 1);             //Variable
        sqlComando.Parameters.AddWithValue("@Valor", Datos.PaxMed[i]);      //Valor
        sqlComando.Parameters.AddWithValue("@Validez", 0);                  //Validez
        sqlComando.Parameters.AddWithValue("@Porcentaje", fPorcentaje);     //Porcentaje
        //Ejecutar la consulta
        sqlComando.ExecuteNonQuery();

        //Borra los parámetros
        sqlComando.Parameters.Clear();


    }
}
```

Figura 5-31. Función *EscribirMedidas*

La función *EscribirAlarmas* **(Figura 5-32)** realiza la gestión de eventos producidos por las alarmas de la estación. Para ello compara los estados de las alarmas recibidos en el paquete de datos del PLC con los estados almacenados en la base de datos. Si el estado ha cambiado, introduce en la tabla eventos de la estación un registro notificando la fecha y el cambio que se ha producido, y modifica el estado almacenado en la base de datos.

```
//Escribir Tabla Eventos de Estaciones y Alarmas de Estaciones
private void EscribirAlarmas(int ER, PaxStruct Datos, SqlCommand sqlComando)
{
    int i;
    String IdER = SaicaId(ER);
    string sComando;
    SqlDataReader sqlUltimoEstado;

    //Evaluar cambios de estados/alarmas (!6 posibles)
    //Comparar datos leidos con datos guardados en la base de datos
    for (i = 0; i < 16; i++)
    {
        //Crear la consulta
        sComando = "SELECT * FROM A" + IdER +" WHERE Id = @Id";
        sqlComando.CommandText = sComando;
        sqlComando.Parameters.AddWithValue("@Id", i + 1);                  //IdAlarma
        //Ejecutar la consulta
        sqlUltimoEstado = sqlComando.ExecuteReader();

        if (Datos.PaxBit[i] != Convert.ToBoolean(sqlUltimoEstado["Estado"])) //Existe cambio de estado
        {
            if (Datos.PaxBit[i]) //Activado
            {
                //Añadir evento
                //Crear la consulta
                sComando = "INSERT INTO E" + IdER + " (Fecha_Hora, IdAlarma, Estado)" +
                    " VALUES (@Fecha_Hora, @IdAlarma, @Estado)";
                sqlComando.CommandText = sComando;
                sqlComando.Parameters.AddWithValue("@Fecha_Hora", Datos.PaxFecha);  //Fecha
                sqlComando.Parameters.AddWithValue("@IdAlarma", i + 1);             //IdAlarma
                sqlComando.Parameters.AddWithValue("@Estado", "Activado");          //Estado
                //Ejecutar la consulta
                sqlComando.ExecuteNonQuery();
            }
            else //Desactivado
            {
                //Añadir evento
                //Crear la consulta
                sComando = "INSERT INTO E" + IdER + " (Fecha_Hora, IdAlarma, Estado)" +
                    " VALUES (@Fecha_Hora, @IdAlarma, @Estado)";
                sqlComando.CommandText = sComando;
                sqlComando.Parameters.AddWithValue("@Fecha_Hora", Datos.PaxFecha);  //Fecha
                sqlComando.Parameters.AddWithValue("@IdAlarma", i + 1);             //IdAlarma
                sqlComando.Parameters.AddWithValue("@Estado", "Desactivado");       //Estado
                //Ejecutar la consulta
                sqlComando.ExecuteNonQuery();

            }
            //Modificar alarma
            //Crear la consulta
            sComando = "UPDATE A" + IdER + " SET Estado = @Estado" +
                " WHERE Id = @Id";
            sqlComando.CommandText = sComando;
            sqlComando.Parameters.AddWithValue("@Estado", Datos.PaxBit[i]);     //Estado
            sqlComando.Parameters.AddWithValue("@Id", i + 1);                   //IdAlarma
            //Ejecutar la consulta
            sqlComando.ExecuteNonQuery();

            //Borra los parámetros
            sqlComando.Parameters.Clear();
        }
    }
}
```

Figura 5-32. Función *EscribirAlarmas*

Cuando el proceso de comunicación con la estación finaliza, .NET Framework se encarga de cerrar el hilo abierto y liberar los recursos que ha utilizado.

5.3 Interfaz de usuario

La interfaz gráfica permite al técnico del centro de control gestionar las comunicaciones con las distintas estaciones y revisar los datos obtenidos de las mismas.

Cuando el programa se inicia, se realiza una comprobación para determinar que no se esté ejecutando ya **(Figura 5-33)**. Si el programa ya se encuentra en ejecución, se detiene la ejecución que ha comenzado nueva. Si se inicia por primera vez, muestra el formulario principal (Form1).

```
namespace SAICA
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            //Comprobar aplicaciones abiertas
            Process[] Aplicaciones = Process.GetProcessesByName(Process.GetCurrentProcess().ProcessName);
            if (Aplicaciones.Length > 1)
            {
                MessageBox.Show("La aplicación ya se encuentra en marcha");
                return;
            }
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Figura 5-33. Inicio de la aplicación

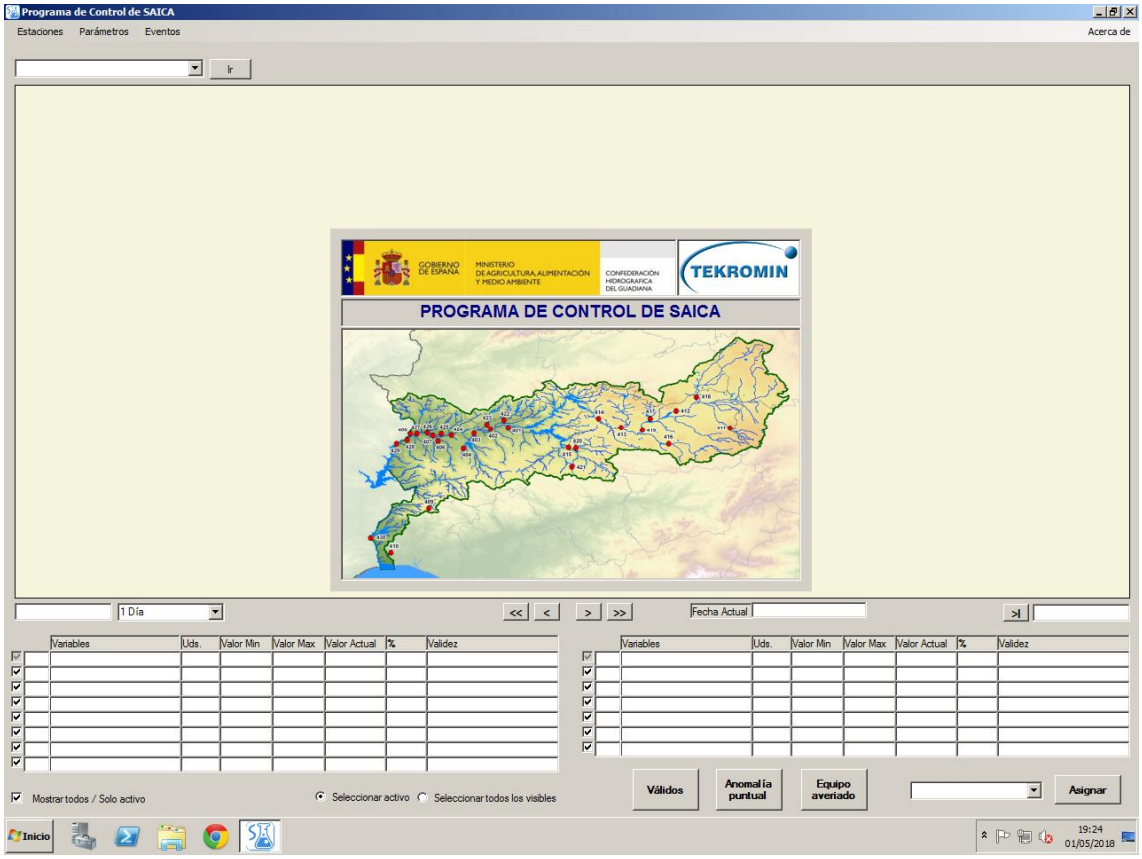

Figura 5-34. Pantalla de carga

Mientras el formulario principal realiza la carga todos los controles, se muestra una pantalla de presentación **(Figura 5-34)**. Una vez acabado el tiempo de muestra se le pide al técnico que se identifique **(Figura 3-35)**. La ventana de identificación sólo bloquea el uso del formulario principal hasta que el usuario introduce sus datos. Esto no impide al programa ejecutar los procesos de comunicación en segundo plano. Cuando se pulsa sobre el botón *Salir*, el formulario se minimiza y el programa sigue ejecutándose.

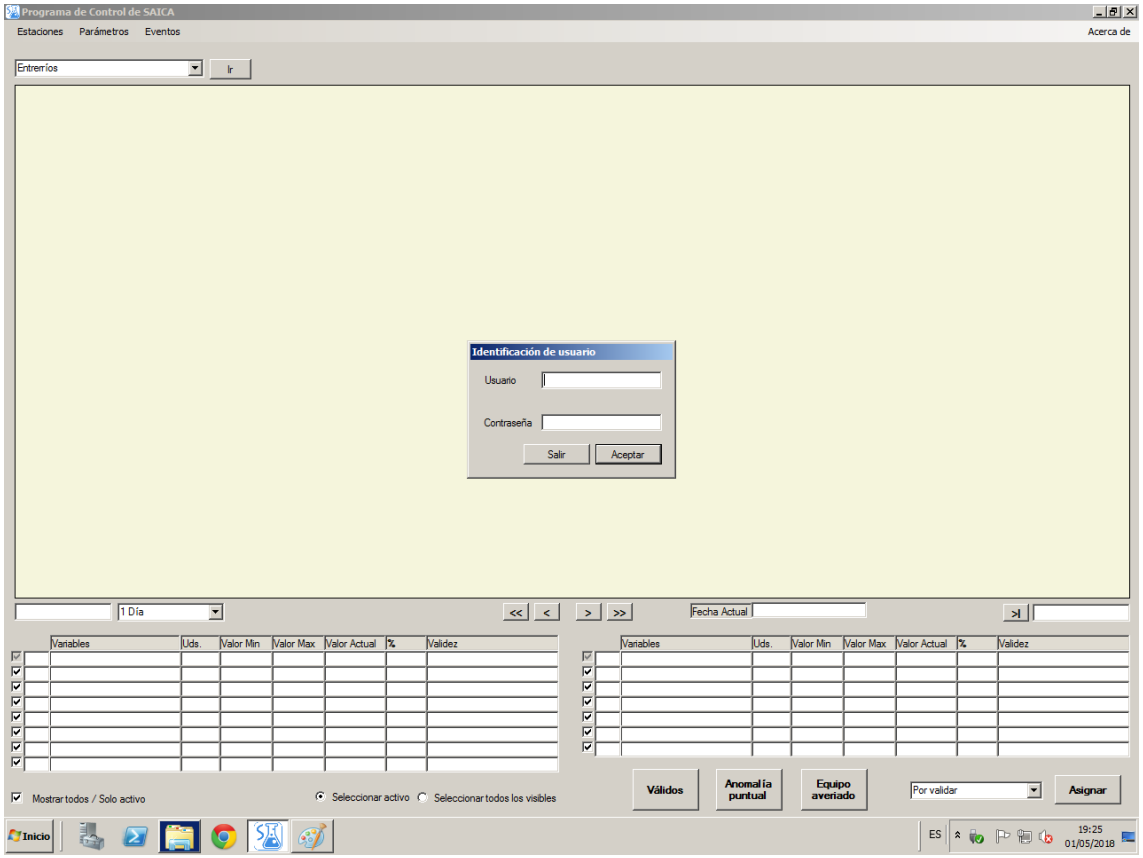

Figura 5-35. Identificación de usuario

El formulario principal está formado por un menú, una sección de selección de estación, la gráfica, una sección de control del eje temporal de la gráfica, una parrilla de información de las variables, y una sección para la validación. Si se cierra la ventana del formulario principal, el programa se minimizará como cuando se pulsa en el botón *Salir* de la identificación de usuario y continúa su ejecución en segundo plano. Para terminar la ejecución o volver a abrir la ventana de nuevo, se deberá hacer click derecho sobre el icono del programa **(Figura 5-36)** en la barra de tareas de Windows y elegir la opción que se desee. Al reabrir la ventana, se volverá a pedir la identificación de usuario.

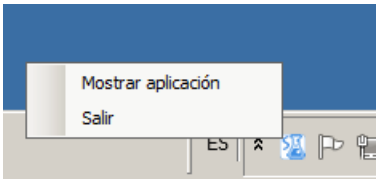

Figura 5-36. Icono de la aplicación en la barra de tareas

La barra de menú se encuentra en la parte superior de la ventana. A través del menú del formulario principal **(Figura 5-37)**, es posible acceder a otros formularios donde el técnico puede gestionar las comunicaciones, preferencias en la visualización de datos o consultar los eventos almacenados en la base de datos.

Figura 5-37. Barra de menú

La zona de selección de estación **(Figura 5-38)** se encuentra bajo la barra de menú. Para seleccionar la estación que se quiere visualizar existe un control *ComboBox* que muestra todas las estaciones de la red. Una vez seleccionada la estación, al pulsar sobre el botón *Ir* se muestra el nombre de la estación elegida y se cargan sus datos.

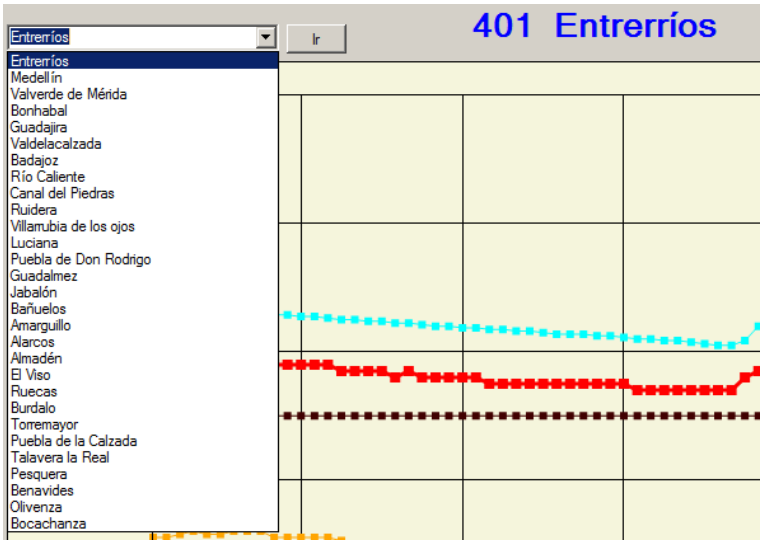

Figura 5-38. Selección de estación

La gráfica **(Figura 5-39)** es el control más importante de la aplicación. Se encuentra en el centro del formulario principal y ocupa gran parte de éste para permitir al usuario una buena visualización de los datos. La gráfica muestra los datos de las variables seleccionadas representando sus valores en unos ejes cartesianos. El eje de abscisas corresponde a las horas y fechas en las que se han tomado los datos de las redes de sensores de las estaciones. Los ejes de ordenadas representan los valores que toman las variables. El eje de la izquierda contiene los valores de la variable resaltada en unidades de ingeniería, y el eje de la derecha representa el valor de todas las variables en porcentaje. Cada variable se representa con un color y la validez de los datos mostrados se indica mediante la forma del punto. Por defecto, un dato por validar se muestra con un cuadrado, un dato no válido (independientemente del motivo) es un aspa y un punto válido es un punto.

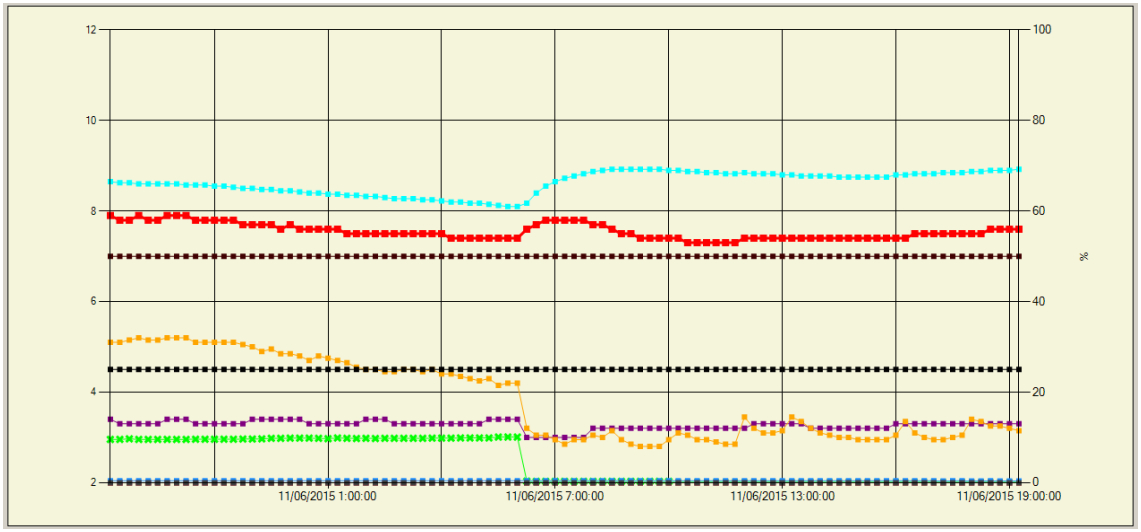

Figura 5-39. Gráfica del programa de validación

En la gráfica se pueden mostrar desde una variable a todas las variables a la vez. En cualquier caso, siempre hay una variable resaltada sobre las demás, representada con un grosor mayor. Esta variable corresponde a la variable destacada en la zona de variables, y no se puede deshabilitar su representación. Tampoco se permite destacar una variable que no esté representada.

Si se hace click en la gráfica, aparece un cursor rojo vertical sobre ella. Este cursor se sitúa siempre sobre un dato existente en la variable. Los valores de los datos sobre el que está el cursor aparecen en la cuadrícula de variables. Si se quiere seleccionar varios datos al mismo tiempo **(Figura 5-40)**, sólo hay que pulsar con el ratón sobre un punto de la gráfica y arrastrar hacia un lado u otro hasta que todos los puntos que se quieran seleccionar aparezcan resaltados con un cuadrado negro.

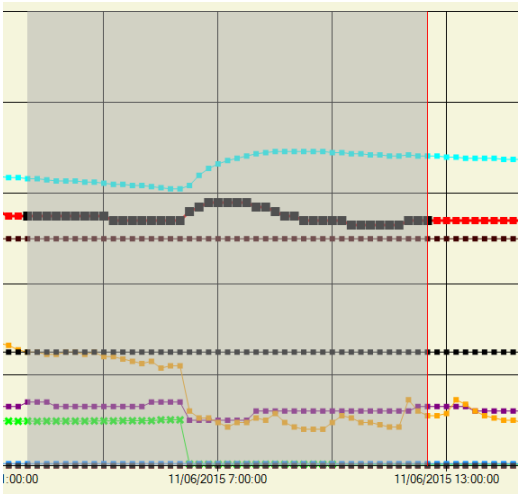

Figura 5-40. Selección de datos sobre la gráfica

El control del eje temporal **(Figura 5-41)** se encuentra justo debajo de la gráfica. Está compuesto por (de izquierda a derecha) el control de fecha inicial de representación, el control de periodo de tiempo representado, el botón de desplazamiento de un periodo hacia atrás en el tiempo, el botón de retroceso de medio periodo, el botón de avance de medio periodo, el botón de avance de un periodo completo, la fecha en la que se encuentra el cursor de la gráfica, el botón de avance hasta la última fecha disponible en la base de datos, y el control de fecha final de representación.

Figura 5-41. Control del eje temporal del programa de validación

La zona donde se muestra la información de las variables **(Figura 5-42)** se encuentra en la parte baja del formulario. Las tablas de las que se compone muestra la información de cada variable en una fila. Las filas de las tablas contienen un control *CheckBox* donde se decide las variables que se muestran en la gráfica, el color con el que se representa la variable, el nombre de la variable, su unidad de ingeniería, el valor mínimo representado, el valor máximo representado, el valor de los datos que señala el cursor de la gráfica, el porcentaje que representa esos valores y la validez de ese dato. Haciendo click sobre cualquier fila, se resalta la variable que esta contiene, cambiando los valores en el eje de ordenadas principal y resaltando también dicha variable en la gráfica. La variable resaltada no se permite deseleccionar, al igual que también se impide resaltar una variable que no esté siendo mostrada en la gráfica. En la parte baja de esta zona se da la opción de representar o dejar de mostrar todas las variables en la gráfica. Obviamente, la variable resaltada no dejará de mostrarse. El control *RadioButton* para la selección de datos en la gráfica permite elegir entre marcar sólo los datos de la variable resaltada dentro de la selección de la gráfica, o marcar todos los datos que aparecen en esta selección.

Figura 5-42. Zona de información de variables del programa de validación

La zona de validación **(Figura 5-43)** aparece en la esquina inferior derecha de la ventana. Consta de tres botones para la validación directa con los valores que más utiliza el técnico. Aparte, contiene un control *ComboBox* para seleccionar la validez que se quiera aplicar para asignarlo pulsando el botón *Asignar*. Cualquiera de los botones que se pulse asigna la validez a los datos que aparecen en la gráfica marcados de color negro.

Figura 5-43. Zona de validación

El formulario de gestión de conexiones a las estaciones **(Figura 5-44)** es accesible pulsando sobre *Estaciones* en el menú del formulario principal y luego haciendo click sobre *Conexiones*. La pantalla de conexiones es la utilizada para activar las conexiones con las estaciones. Para activar/desactivar la conexión se han de habilitar/deshabilitar las casillas de la columna ñActivoò correspondientes a las estaciones que se deseen modificar. Una vez realizadas las modificaciones, al pulsar sobre el botón *Aplicar* o el botón *Aceptar* se registrarán los cambios en el sistema.

En la aplicación SAICA, existe la posibilidad de elegir la conexión preferente a cada estación. Puede ser vía VPN o TETRA.

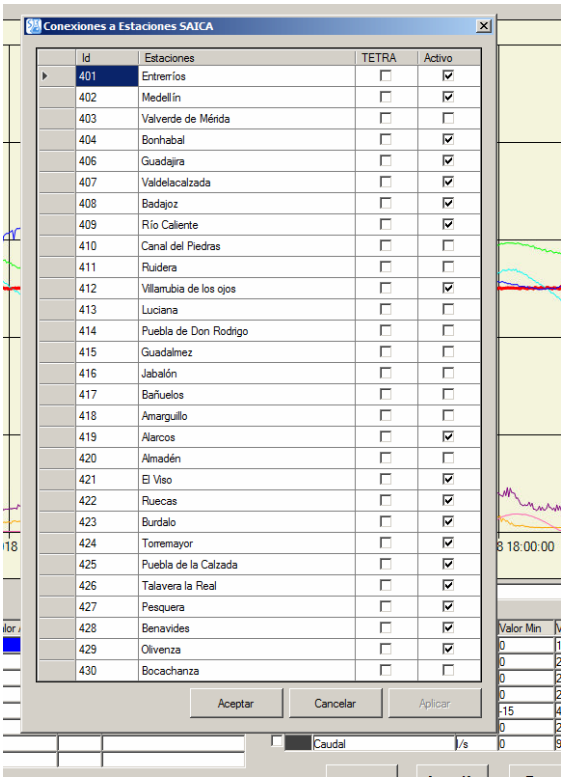

Figura 5-44. Formulario de Conexiones a estaciones SAICA

Para acceder al formulario de direcciones IP de las estaciones **(Figura 5-45)** se pulsa sobre *Estaciones* en el menú del formulario principal y después en *Direcciones*. En la pantalla de direcciones IP se puede consultar y modificar el direccionamiento que poseen las estaciones del sistema, aparte de mostrarse otras informaciones sobre las estaciones.

En el caso de la aplicación para SAICA, se muestra el nombre de la estación junto al río y el municipio en el que se encuentra. En la parte derecha de la cuadrícula aparecen las direcciones IP para la conexión vía VPN o vía TETRA.

Para la aplicación REACAR, la información disponible es, junto al nombre de estación, si pertenece al sistema REACAR Oriental u Occidental y la dirección IP de la única vía de conexión (VPN).

Para introducir un cambio en las direcciones IP de las estaciones, el formato de la dirección introducida tiene que ser correcto (xxx.xxx.xxx.xxx). Una vez introducidos los cambios, puede pulsar el botón *Aplicar* o el botón *Aceptar* para introducir los cambios en el sistema. Si el formato de alguna dirección es incorrecto, aparecerá un mensaje de aviso y no se introducirán los cambios en el sistema.

Para salir de la pantalla sin aplicar ningún cambio, simplemente se ha de pulsar el botón *Cancelar*.

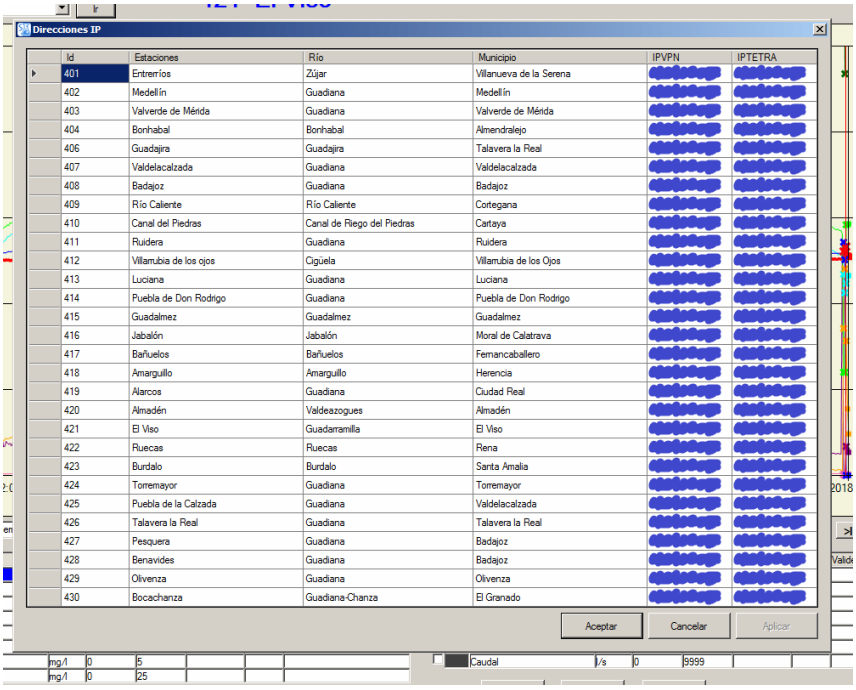

Figura 5-45. Formulario de Direcciones IP de la aplicación de la red SAICA

Si se quiere abrir el formulario donde se gestionan las variables que se muestran por defecto en la gráfica **(Figura 5-46)**, hay que pulsar sobre *Parámetros* en el menú y luego en *Variables*.

Cada vez que se elige en la pantalla principal una nueva estación, se muestran en la gráfica una serie de variables por defectos. Para seleccionar qué variables se muestran por defecto en cada estación, se utiliza la ventana de variables.

En la ventana de variables aparece una tabla con los nombres de las variables y una casilla a la derecha de cada variable donde se elige si la variable se muestra por defecto o no. Para elegir la estación que se desea modificar, se abre la pestaña con el identificador de estación. Cada estación tiene dentro de su pestaña su propia tabla de variables.

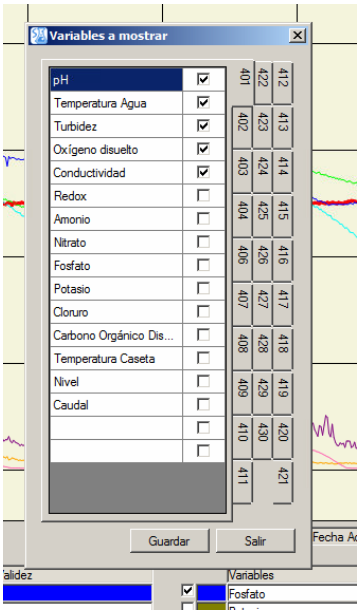

Figura 5-46. Formulario de Variables a mostrar de la aplicación de la red SAICA

Una vez realizadas las modificaciones en las tablas de las estaciones, se pulsa sobre el botón *Guardar* y los cambios serán introducidos en el sistema. No es posible dejar una estación sin variables a mostrar por defecto,

así pues, si se ha dejado alguna estación sin ninguna variable activada, aparecerá un error en el momento de guardar y los cambios no serán introducidos en el sistema.

La edición de los parámetros de validez se realiza desde el formulario de motivos de invalidez **(Figura 5-47)**. Este formulario se muestra pulsando sobre *Parámetros* en el menú de la pantalla principal y a continuación en *Validez*.

En la ventana de motivos de invalidez, se pueden añadir, modificar o eliminar valores de validez. Los únicos no modificables ni eliminables son los cuatro primeros valores.

Para añadir un nuevo motivo de invalidez, hay que pulsar sobre el botón *Añadir* y se abrirá una nueva ventana donde se introducirá el texto de invalidez. Una vez guardado, aparecerá en la tabla junto a los demás y en el combo en la pantalla principal.

El método para modificar un motivo de invalidez es seleccionar ese valor en la tabla y pulsar sobre el botón *Editar*. Aparecerá una ventana donde introducir el motivo. Una vez guardado, el nuevo texto aparecerá en el valor seleccionado.

Para eliminar algún valor de invalidez, se selecciona el valor que se quiere eliminar y se pulsa sobre el botón *Eliminar*. El valor será borrado automáticamente y desaparecerá de la tabla y del combo de la pantalla principal.

El programa impide modificar las cuatro primeras filas.

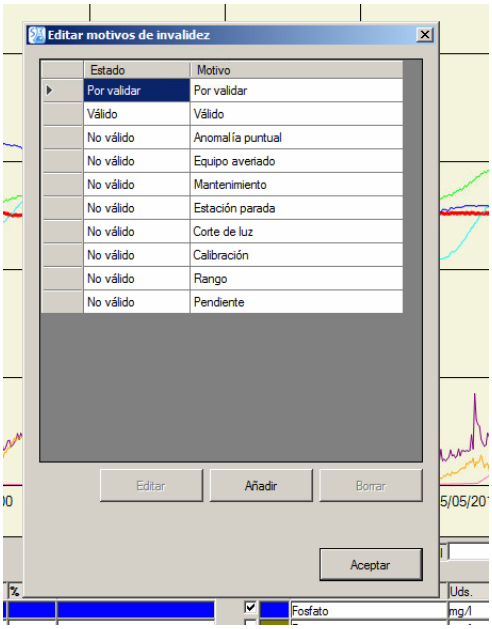

Figura 5-47. Formulario de Motivos de invalidez

Desde el formulario de eventos de comunicación **(Figura 5-48)** es posible consultar los eventos producidos durante las comunicaciones con las estaciones. El formulario se abre desde la opción *Eventos* y *Comunicaciones* del menú del formulario principal. Aquí se informa de la estación que ha fallado, la hora del error, el error registrado y el tipo de error que es.

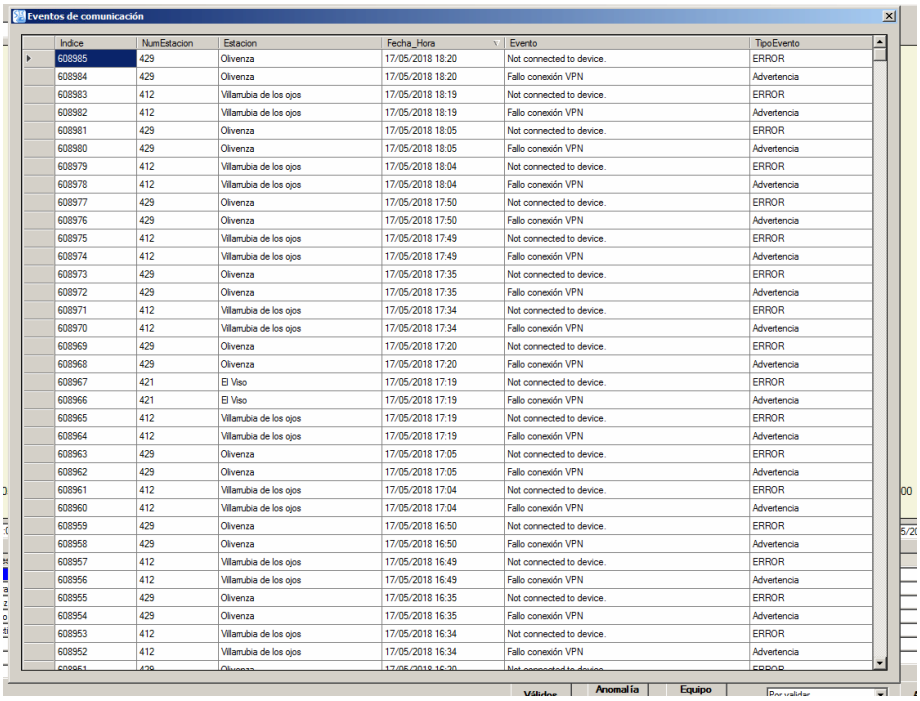

Figura 5-48. Formulario de Eventos de comunicación

Los eventos producidos por las alarmas de las estaciones se consultan desde el formulario eventos de estación **(Figura 5-49)**. Éste es accesible pulsando sobre *Eventos* en el menú de la ventana principal para después hacer click sobre la opción *Estaciones*. Dentro del formulario, se selecciona la estación de la que se quiere consultar la información en un control *ComboBox* y pulsando sobre el botón *Ir*.

Si se quiere modificar la descripción de las alarmas, esto se realiza desde el formulario que se abre pulsando sobre el botón *Eventos*.

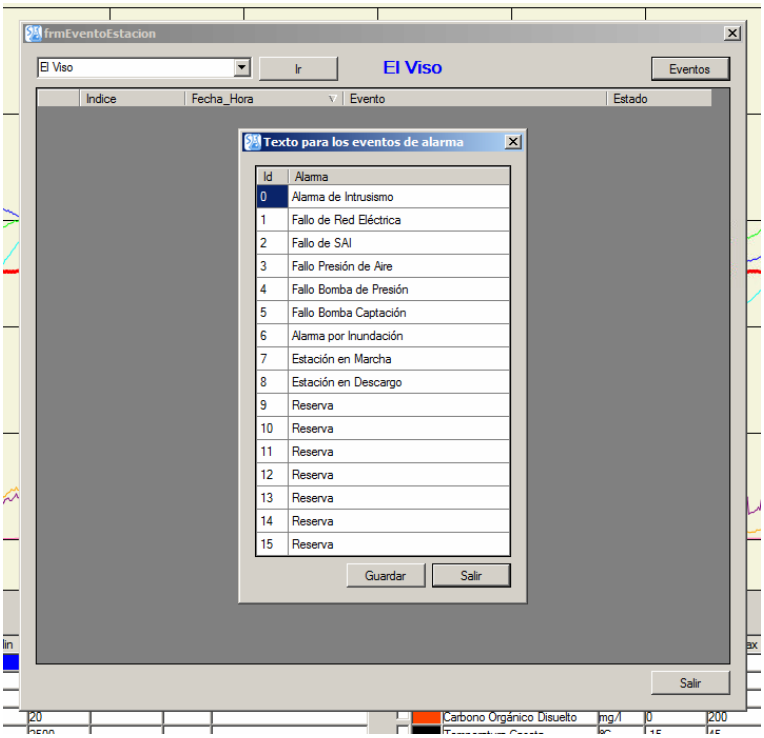

Figura 5-49. Formulario de Eventos de estación

5.3.1.1 Funcionamiento de la interfaz de usuario

En esta sección se explican los procesos que efectúa la aplicación cuando el usuario actúa sobre la interfaz gráfica. Los controles que contienen los formularios generan eventos en tiempo de ejecución que inician funciones dentro del programa. El uso principal que le da un usuario a la aplicación es la validación de datos de las estaciones de medida. El proceso de validación es el explicado en el apartado **3.1 Diseño del sistema (Figura 3-7)**. A continuación, se expone cómo el programa procede con ello.

El programa utiliza varias variables globales dentro de cada formulario que son accesibles desde todas las funciones ejecutadas en su formulario correspondiente. En el formulario principal, las variables globales son las mostradas en la figura **(Figura 5-50)**.

```
public int foco = 0; //Dato a resaltar en la gráfica
public string IdEstacion = ""; //Identificador de la estación que se representa
public bool pintando = false; //Nos indica cuando se ha acabado de representar los datos
public bool marcando = false; //Nos indica cuando se ha acabado de representar los datos
public DateTime dateIni, dateFin; //Fecha Inicial y Final que se representa en la gráfica
public bool checkCambio = true; //Deshabilita el evento cbMostrarTodo_CheckedChanged
public bool checkVarMed = true; //Deshabilita el evento checkVar_CheckedChanged
public bool selMostrarTodo = false; //Indica si se valida solo el foco(false) o todas la variables visibles(true)
public bool timerActivo = false;
```

Figura 5-50. Variables globales de Form1

Cuando el programa abre por primera vez el formulario principal, carga los objetos que se han creado durante el tiempo de diseño. Estos objetos son:

- Controles mostrados en pantalla. Estos controles ya han sido nombrados.
- Objetos *TableAdapter* **(Figura 5-51)**. Se crean *TableAdapter* relacionados con las tablas de la base de datos a las que se accede desde el formulario. El funcionamiento de estos objetos viene explicado en el apartado **3.2 Base de datos**.

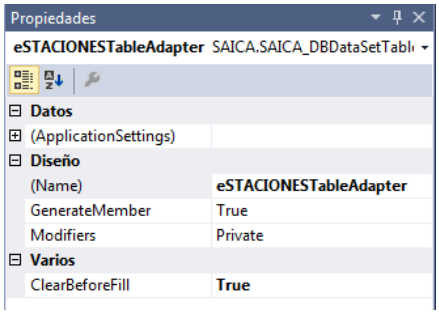

Figura 5-51. Propiedades de un *TableAdapter*

- Objetos *BindingSource* **(Figura 5-52)**. Estos objetos se utilizan para enlazar controles a objetos *DataTable*. Los objetos *DataTable* que son enlazadas en este formulario pertenecen a *sAICA_DBDataSet*. Permite filtrar las filas de la tabla que se quieren enlazar.

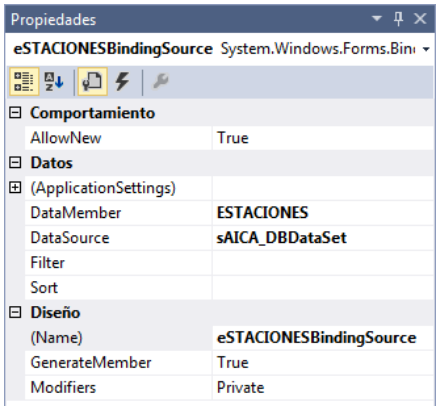

Figura 5-52. Propiedades de un *BindingSource*

- Objeto *Timer* **(Figura 5-53)**. Es utilizado para generar el evento que inicia el proceso de lectura de las estaciones.

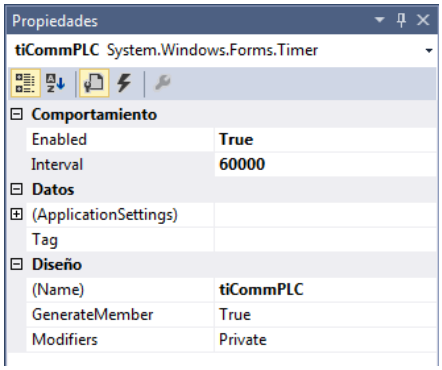

Figura 5-53. Propiedades de un *Timer*

Una vez se termina de cargar el formulario, lo primero que debe hacer el técnico para realizar una validación es seleccionar una estación en el control *ComboBox* de la zona de selección de estación y pulsar sobre *Ir*. Cuando se hace click sobre el botón *Ir*, se genera un evento que llama a la función *buttonEstacion_Click*.

La función *buttonEstacion_Click* empieza el proceso de lectura de datos de la base de datos. Los pasos dados por esta función son los siguientes:

- Habilita los controles inactivos del formulario.
- Toma la estación seleccionada en el *ComboBox* y muestra el código y el nombre de la estación sobre la gráfica **(Figura 5-54)**.

```
if (comboEstacion.SelectedValue != null)
{
    //Toma el valor del combo de Estaciones y muestra en pantalla que estación se está representando
    selIndexEstacion = comboEstacion.SelectedIndex;

    IdEstacion = sAICA_DBDataSet.ESTACIONES[selIndexEstacion].Id;
    Estacion = sAICA_DBDataSet.ESTACIONES[selIndexEstacion].Estaciones;

    richTextBoxNombreEstacion.Clear();
    richTextBoxNombreEstacion.Text = IdEstacion + "  " + Estacion;
```

Figura 5-54. Lectura del código y nombre de la estación

- Selecciona las variables a mostrar por defecto llamando a la función *CargaChecks*.
- Lee la fecha del último dato almacenado en la tabla de la estación.
- Si la tabla contiene datos, calcula la fecha inicial a representar **(Figura 5-55)**.

```
fin = sAICA_DBDataSet.Tables["S" + IdEstacion].Rows.Count;
if (fin != 0) //Si existen datos en la tabla de estación...
{
    time = Periodo();
    //Se toma la última fecha en la que hay dato y se introduce el periodo que ha de representarse
    DataRow row = sAICA_DBDataSet.Tables["S" + IdEstacion].NewRow();
    row = sAICA_DBDataSet.Tables["S" + IdEstacion].Rows[fin - 1];
    dateFin = Convert.ToDateTime(row["Fecha_Hora"]);

    if (time == TimeSpan.FromSeconds(0))
    {
        if (dateFin.Month == 1)
            dateIni = new DateTime(dateFin.Year - 1, 12, dateFin.Day,
                dateFin.Hour, dateFin.Minute, dateFin.Second);
        else
            dateIni = new DateTime(dateFin.Year, dateFin.Month - 1, dateFin.Day,
                dateFin.Hour, dateFin.Minute, dateFin.Second);
    }
    else
        dateIni = dateFin.Subtract(time);
```

Figura 5-55. Cálculo de fechas de representación

  - Llama a la función *PintaGrafica*.
- Si la tabla no contiene datos, pone las fechas a cero.
  - Llama a la función *PintaGrafica*.
- Resetea los valores del cursor

La función *PintaGrafica* es la encargada de representar los datos en la gráfica. La gráfica es el control *Chart1*. Un control *Chart* está formado por un grupo de propiedades y objetos que hay que configurar para que las representaciones aparezcan correctamente. La función procede así:

- Se toma el control *BindingSource* de la estación a representar.
- Se filtra el *BindingSource* y se asocia la gráfica **(Figura 5-56)**. Hay que destacar este paso puesto que es importantísimo para que el funcionamiento del programa sea fluido. Sin filtrar el control *BindingSource* la gráfica cargaría todos los datos contenidos en la tabla. Esto conllevaría que conforme fuese pasando el tiempo y las tablas de datos fuesen creciendo, el programa llegaría a ser inmanejable puesto que tendría que usar muchísimos recursos del equipo.

```
if (BsActivo != null)
{
    BsActivo.RemoveFilter();
    BsActivo.Filter = "Fecha_Hora >= '" + dateIni.ToString() +
        "' AND Fecha_Hora <= '" + dateFin.ToString() + "'";

    //Seleccionamos el origen de dato del chart1
    chart1.DataSource = BsActivo;
```

Figura 5-56. Enlace de la gráfica con la estación

- Configura el eje de abscisas del área de representación **(Figura 5-57)**.

```
//Representación en el eje X
textBoxFechaFin.Text = dateFin.ToString(); //Se introduce valor final en el textBox
textBoxFechaIni.Text = dateIni.ToString(); //Se introduce valor inicial en el textBox
chart1.ChartAreas["ChartArea1"].AxisX.Minimum = dateIni.ToOADate();
chart1.ChartAreas["ChartArea1"].AxisX.Maximum = dateFin.ToOADate();
chart1.ChartAreas["ChartArea1"].AxisX.Interval = time.TotalMinutes / 4;
chart1.ChartAreas["ChartArea1"].AxisX.MajorGrid.Interval = time.TotalMinutes / 8;
pintando = true;
```

Figura 5-57. Configuración eje X.

- Comienza la representación de datos **(Figura 5-58)**.
    - Comprueba si la variable debe ser representada.

```
//Comienza la representación de dato
for (i = 1; i < 16; i++)
{
    serie = "Series" + i.ToString();

    check = ComprobarCheck(i);
    if (check == true) //Comprobación de si ha de representarse esta serie
    {
```

Figura 5-58. Comprobación de representación

    - Crea la serie en caso de no existir. Una serie es un grupo de puntos a representar conjuntamente. Se representa la serie con una línea **(Figura 5-59)**.
    - Asigna el campo *Fecha_Hora* al eje de abscisas **(Figura 5-59)**.
    - Si la variable tiene el foco, asigna los campos *Valor*, *Validez* y *Variable* al eje de ordenadas **(Figura 5-59)**.
        - Configura el eje de ordenadas en el área de representación.
        - Da color y grosor a la representación de la serie y la asocia al eje Y principal.
    - Si la variable no posee el foco, se asignan los campos *Porcentaje*, *Validez* y *Variable* al eje de ordenadas **(Figura 5-59)**.
        - Da color y grosor a la serie y la asocia al eje Y secundario.

```
//Si la serie no existe, se crea
if (chart1.Series.IsUniqueName(serie))
{
    chart1.Series.Add(serie);
    chart1.Series[serie].ChartType =
        System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
}
//Selección del valor a representar en el eje X
chart1.Series[serie].XValueMember = "Fecha_Hora";
chart1.Series[serie].YValuesPerPoint = 3;
if (!chart1.Series.IsUniqueName(serie))
{
    if (i == foco) //Si la serie actual es de la variable seleccionada, se resalta
    {
        chart1.Series[serie].ChartArea = "ChartArea1";

        //Selección de datos del eje Y. Se muestra el valor
        chart1.Series[serie].YValueMembers = "Valor, Validez, Variable";
        chart1.ChartAreas["ChartArea1"].AxisY.Minimum = sAICA_DBDataSet.VARIABLES.FindById(i).V_Min;
        chart1.ChartAreas["ChartArea1"].AxisY.Minimum = sAICA_DBDataSet.VARIABLES.FindById(i).V_Max;
        chart1.ChartAreas["ChartArea1"].AxisY.Title = sAICA_DBDataSet.VARIABLES.FindById(i).Ud_Ing;

        //Elección de color, grosor y eje de la serie en la que se encuentra el foco
        chart1.Series[serie].Color = chart1.PaletteCustomColors[i - 1];
        chart1.Series[serie].BorderWidth = 3;
        chart1.Series[serie].YAxisType =
            System.Windows.Forms.DataVisualization.Charting.AxisType.Primary;
        chart1.ChartAreas["ChartArea1"].AxisY2.MajorGrid.Enabled = false;
    }
    else //Si la serie no posee el foco, se representa en el eje Y secundario
    {
        //Selección de los datos del eje Y2. Se muestra el porcentaje
        chart1.Series[serie].YValueMembers = "Porcentaje, Validez, Variable";

        //Elección de color, grosor y eje
        chart1.Series[serie].YAxisType =
            System.Windows.Forms.DataVisualization.Charting.AxisType.Secondary;
        chart1.Series[serie].ChartArea = "ChartArea1";
        chart1.Series[serie].Color = chart1.PaletteCustomColors[i - 1];
        chart1.Series[serie].BorderWidth = 1;
    }
```

Figura 5-59. Configuración de una serie

- Filtra los puntos de la serie para que solo se representen los datos de la variable que corresponda **(Figura 5-60)**.

```
//Filtrado por variable a mostrar. Mostrar en la serie la varieble i
chart1.DataManipulator.FilterMatchedPoints = false;
chart1.DataManipulator.Filter(CompareMethod.EqualTo, i, serie, serie, "Y3");
```

Figura 5-60. Filtrado de puntos de la serie por variable

- Configura el eje de ordenadas secundario y se fuerza que la serie sea representada mediante una fila **(Figura 5-61)**.

```
//Datos a mostrar en el eje Y secundario
chart1.ChartAreas["ChartArea1"].AxisY2.Minimum = 0;
chart1.ChartAreas["ChartArea1"].AxisY2.Maximum = 100;
chart1.ChartAreas["ChartArea1"].AxisY2.Title = "%";
chart1.Series[serie].ChartType =
    System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Column;
chart1.Series[serie].ChartType =
    System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
```

Figura 5-61. Configuración del eje Y secundario

- Carga los valores correspondientes en las tablas de variables.

Justo antes de dibujar la imagen del gráfico, se desencadena el evento Customize. Este evento llama a la función *PintaPuntos* a no ser que ya haya sido llamada.

La función *PintaPuntos* dibuja los marcadores de los puntos en función de su validez. Los datos que aún están por validar se representan mediante un punto cuadrado, los datos inválidos se muestran con un aspa y los datos válidos no tienen marcador, de esta forma sólo se percibe la línea de la serie. La función se muestra en la figura **(Figura 5-62)**.

- La función comienza tomando los puntos de inicio y final de la selección realizada en la gráfica (en el caso de haberla) .
- Realiza la representación de puntos uno a uno.

- Si el punto se encuentra dentro de la selección, los marcadores de los puntos tienen forma cuadrada, de color negro y tamaño 10.
- Si están fuera de la selección, se pintan según su validez.
  - Si su estado está sin validar, se representa con un cuadrado del mismo color que la línea de su variable y de tamaño 8 si la variable tiene el foco y 6 si no lo tiene.
  - Si el dato es válido, no muestra marcador, simplemente la línea pasa por ese punto.
  - Si el dato no es válido, el marcador es un aspa del mismo color que la línea y de tamaño 12 si la variable tiene el foco y 10 si no lo tiene.

- Finalmente, fuerza la representación de la serie como una línea de puntos.

```
public void PintaPuntos(string serie, int ID)
{
    double xIni, xFin;
    DataPoint dataPoint, dataPoint0, dataPoint1, dataPoint2;//Puntos de la serie: Por validar, Válidos, No válidos

    try
    {
        xIni = chart1.ChartAreas["ChartArea1"].CursorX.SelectionStart;
        xFin = chart1.ChartAreas["ChartArea1"].CursorX.SelectionEnd;
        if (!chart1.Series.IsUniqueName(serie))
        {
            //Se comprueba la validez de los puntos de la serie
            foreach (DataPoint dp in chart1.Series[serie].Points)
            {
                if (((dp.XValue <= xFin && dp.XValue >= xIni) || (dp.XValue <= xIni && dp.XValue >= xFin))
                    && Button.MouseButtons != MouseButtons.Left)
                {
                    //Puntos dentro de la selección
                    dataPoint = dp;
                    dataPoint.MarkerColor = Color.Black;
                    dataPoint.MarkerStyle = MarkerStyle.Square;
                    dataPoint.MarkerSize = 10;
                }
                else
                {
                    if (dp.YValues[1].ToString() == "0")
                    {
                        //Si la validez del dato es 0, se representa un cuadrado
                        dataPoint0 = dp;
                        dataPoint0.MarkerStyle = MarkerStyle.Square;
                        dataPoint0.MarkerColor = chart1.PaletteCustomColors[ID - 1];
                        if (ID == foco)
                            dataPoint0.MarkerSize = 8;
                        else
                            dataPoint0.MarkerSize = 6;
                    }
                    else if (dp.YValues[1].ToString() == "1")
                    {
                        //Si la validez es 1, no se representa con marcador
                        dataPoint1 = dp;
                        dataPoint1.MarkerSize = 1;
                        dataPoint1.MarkerStyle = MarkerStyle.None;
                        dataPoint1.MarkerColor = chart1.PaletteCustomColors[ID - 1];
                    }
                    else
                    {
                        //Si el dato no es válido, se representa con un aspa
                        dataPoint2 = dp;
                        dataPoint2.MarkerStyle = MarkerStyle.Cross;
                        dataPoint2.MarkerColor = chart1.PaletteCustomColors[ID - 1];
                        if (ID == foco)
                            dataPoint2.MarkerSize = 12;
                        else
                            dataPoint2.MarkerSize = 10;
                    }
                }
            }


            chart1.Series[serie].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Column;
            chart1.Series[serie].ChartType = System.Windows.Forms.DataVisualization.Charting.SeriesChartType.Line;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Fallo al mostrar validez" + Environment.NewLine + ex.Message, "Advertencia",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

Figura 5-62. Función *PintaPuntos*

Cuando la función *PintaPuntos* termina, el evento que lo ha llamado finaliza mostrando la gráfica con los valores representados.

Para realizar la selección de los datos que se quieren valorar, en primer lugar se utilizan los controles del eje temporal, se selecciona las variables que se quieren mostrar en la gráfica y se resalta la variable que se quiere valorar. Si se quieren valorar todas las variables mostradas o sólo la que tiene el foco, se selecciona en el control *RadioButton* que hay bajo las tablas de variables.

Los controles de tiempo producen cada uno un evento donde se modifican la fecha de inicio y de final de representación y se llama a la función *PintaGráfica* para que muestre en la gráfica los nuevos datos. Un ejemplo es la función *textBoxFechaIni_KeyDown* **(Figura 5-63)** que es llamada por el evento *KeyDown* cuando se pulsa una tecla del teclado dentro del control de fecha inicial.

```
private void textBoxFechaIni_KeyDown(object sender, KeyEventArgs e)
{
    DateTime date;
    TimeSpan time;

    if (IdEstacion != "")
    {
        if (e.KeyCode == Keys.Return)
        {
            if (DateTime.TryParse(textBoxFechaIni.Text, out date)) //Si el formato es correcto, modifica el intervalo mostrado
            {
                dateIni = Convert.ToDateTime(textBoxFechaIni.Text);

                time = Periodo();

                if (time == TimeSpan.FromSeconds(0))
                    dateFin = dateIni.AddMonths(1);
                else
                    dateFin = dateIni.Add(time);

                PintaGrafica();
            }
            else //Si el formato es incorrecto, muestra un mensaje de error
            {
                textBoxFechaIni.Text = dateIni.ToString();
                string texto = "Formato de Fecha Incorrecto";
                texto += Environment.NewLine;
                texto += "  DD/MM/AAAA hh:mm:ss";
                MessageBox.Show(texto, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }
}
```

Figura 5-63. Función *textBoxFechaIni_KeyDown*

Los controles de selección de variables realizan una función parecida a los controles de tiempo. Esto es cambiar una propiedad o variable del programa y llamar a la función *PintaGrafica*.

Una vez se tiene la gráfica en el periodo que se desea, con las variables que se desean, se procede a la selección de los puntos que se quieren valorar. Para ello se pulsa la tecla izquierda del ratón sobre el lugar donde se quiere comenzar la selección y se arrastra el ratón hasta el punto final. Cuando se suelta la tecla del ratón, se produce un evento que llama a la función *chart1_MouseUp*.

La función *chart1_MouseUp* **(Figura 5-64)** comprueba si se ha de seleccionar sólo los puntos de la variable que tiene el foco, o si se seleccionan todas las variables representadas. La selección de los puntos de una serie la realiza la función *SeleccionarPuntos*.

```
private void chart1_MouseUp(object sender, MouseEventArgs e)
{
    string serie;
    int s;
    if (selMostrarTodo) //Selecciona todos las series visibles en la gráfica
    {
        for (s=1;s<16;s++)
        {
            serie = "Series" + s.ToString();
            if (ComprobarCheck(s))
            {
                SeleccionarPuntos(serie, s);
            }
        }
    }
    else //Selecciona solo la variable seleccionada
    {
        serie = "Series" + foco.ToString();
        SeleccionarPuntos(serie, foco);
    }
}
```

Figura 5-64. Función *chart1_MouseUp*

La función *SeleccionarPuntos* coloca el cursor de la gráfica y el fin de la selección en el punto más cercano **(Figura 5-65)** de donde se ha soltado la tecla del ratón para dar más coherencia a los datos mostrados en los campos de valores actuales. Después llama a la función *PintaPuntos*.

```
x = chart1.Series[serie].Points[0].XValue;
dif1 = Math.Abs(pos - x);

//bucle que recorre la serie
while (i < fin)
{
    x = chart1.Series[serie].Points[i].XValue;
    dif2 = Math.Abs(pos - x);
    //Si la distancia con el cursor es menor que la mínima almacenada se toma una nueva distancia mínima
    if (dif2 < dif1)
    {
        dif1 = dif2;
        index = i;
    }

    i++;
}
x = chart1.Series[serie].Points[index].XValue; //posicion del punto más cercano al cursor
chart1.ChartAreas["ChartArea1"].CursorX.Position = x; // se recoloca el cursor al punto más cercano
chart1.ChartAreas["ChartArea1"].CursorX.SelectionEnd = x; // se recoloca el fin de la selección al punto más cercano
```

Figura 5-65. Cálculo de punto más cercano

Cuando se tienen los puntos de la gráfica ya elegidos, se asigna un estado de validez con los controles de validación. Todos los botones de la zona de validación llaman a la función *SelValidar* **(Figura 5-66)**. Esta función recibe como parámetro la validez a introducir en la base de datos. Los botones *Válido*, *Anomalía puntual* y *Equipo averiado* establecen los valores *1*, *2* y *3* respectivamente, mientras que el botón *Asignar* introduce el valor definido por el control *ComboBox*.

```
public void SelValidar(int ParamValida)
{
    int s;
    if (selMostrarTodo)
    {
        for (s = 1; s < 16; s++)
        {
            if (ComprobarCheck(s))
            {
                Validar(ParamValida, s);
            }
        }
    }
    else
    {
        Validar(ParamValida, foco);
    }
    chart1.ChartAreas["ChartArea1"].CursorX.SelectionStart = 0;
    chart1.ChartAreas["ChartArea1"].CursorX.SelectionEnd = 0;
}
```

Figura 5-66. Función *SelValidar*

La función *SelValidar* repasa las variables que se han valorado, y llama a la función *Validar* **(Figura 5-67)** que es la que modifica los valores en la base de datos.

```
public void Validar(int ParamValida, int s)
{
    string serie = "Series" + s.ToString();
    double xIni, xFin, x;
    int i = 1;
    int fin;

    string sConexion = "SERVER=SAICA-SERVER;DATABASE=SAICA_DB;Trusted_Connection=true";
    string sComando = "UPDATE S" + IdEstacion + " SET Validez = @Validez " +
        "WHERE Fecha_Hora = @Fecha_Hora AND Variable = @Variable";

    try
    {
        if (!chart1.Series.IsUniqueName(serie))
        {
            using (SqlConnection sqlConexion = new SqlConnection(sConexion))
            {
                sqlConexion.Open();
                SqlCommand sqlComando = sqlConexion.CreateCommand();
                sqlComando.CommandText = sComando;
                sqlComando.Connection = sqlConexion;

                xIni = chart1.ChartAreas["ChartArea1"].CursorX.SelectionStart;
                xFin = chart1.ChartAreas["ChartArea1"].CursorX.SelectionEnd;
                fin = chart1.Series[serie].Points.Count;

                for (i = 0; i < fin; i++)
                {
                    x = chart1.Series[serie].Points[i].XValue;
                    if ((x <= xFin && x >= xIni) || (x <= xIni && x >= xFin))
                    {
                        sqlComando.Parameters.AddWithValue("@Validez", ParamValida);
                        sqlComando.Parameters.AddWithValue("@Fecha_Hora", DateTime.FromOADate(x));
                        sqlComando.Parameters.AddWithValue("@vARIABLESBindingSource", s);
                        //Ejecuta la consulta
                        sqlComando.ExecuteNonQuery();

                        //Borra los parámetros
                        sqlComando.Parameters.Clear();
                    }
                }
            }
            PintaPuntos(serie, s);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Fallo al realizar el cambio de validez" + Environment.NewLine + ex.Message,
            "Advertencia", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
```

Figura 5-67. Función *Validar*

Y con ello termina la explicación del proceso de validación. Si se desea continuar validando datos, se repite las partes de este proceso que se necesiten.

La funcionalidad de los otros formularios es reducida y poco interesante puesto que se dedican a leer o modificar en la base de datos de forma similar a lo explicado en el apartado **3.3.1 Comunicación con la base de datos**. Por lo que explicar el código de cada formulario sería repetitivo y poco didáctico.

6 PROGRAMA DE VISUALIZACIÓN WEB

La consulta de datos a través de la red se realiza a través de la navegación Web. Con cualquier navegador Web se puede acceder a los servidores de las redes de seguimiento y visualizar los datos que se ofrecen. Cada servidor ofrece los datos de su red, pero el programa que realiza la consulta a la base de datos y muestra los datos es el mismo para ambos. La única diferencia viene dada por el número y nombre de las estaciones y por la cantidad de variables representadas.

El programa está desarrollado sobre ASP.NET, separando la programación estática de la dinámica en distintos archivos y distintos lenguajes de programación. El código estático es el encargado del diseño gráfico de los formularios utilizando HTML, mientras que en el código dinámico se desarrollan las funciones que llaman los eventos cuando se utiliza cualquier control del formulario.

La explicación del funcionamiento de esta aplicación Web está dividida en tres apartados: la declaración de controles en HTML, la comunicación con la base de datos y la interfaz de usuario.

6.1 Declaración de controles en código estático

Los controles de servidor Web ASP.NET son objetos de páginas web ASP.NET que se ejecutan cuando se solicita la página y que se representan en un explorador. Muchos controles de servidor web son similares a elementos HTML conocidos, como botones y cuadros de texto. Otros controles abarcan comportamiento complejo, por ejemplo controles de gráficas y controles que administran las conexiones de datos.

En este proyecto se utilizan controles de servidor HTML y controles de servidor Web.

Los controles de servidor HTML son elementos HTML que contienen atributos que los convierten en programables en código del servidor. De forma predeterminada, los elementos HTML en una página Web ASP.NET no están disponibles para el servidor. En su lugar, se tratan como texto opaco y se pasan al explorador. Sin embargo, cuando se convierten en controles de servidor HTML, los elementos HTML quedan expuestos como elementos programables en el servidor.

Cualquier elemento HTML de una página se puede convertir en control de servidor HTML agregando el atributo *runat="server"*. Durante el análisis, el marco de trabajo de la página ASP.NET crea instancias de todos los elementos que contienen el atributo *runat="server"*. Si se desea hacer referencia al control como un miembro dentro del código, también se le deberá asignar un atributo *id* al control.

Los controles de servidor Web son un segundo conjunto de controles diseñado con otro enfoque. No se asignan necesariamente uno a uno a controles de servidor HTML. En lugar de ello, se definen como controles abstractos, en los que el marcado real representado por el control puede ser muy diferente al modelo con respecto al que se han programado. Los controles utilizan una sintaxis como la que se muestra a continuación **(Figura 6-1)**:

```
<asp:Button id="btnIzq2" Text="<<" OnClick="btnIzq2_Click" runat="server" />
```

Figura 6-1. Ejemplo de sintaxis de un control de servidor Web

El ejemplo anterior es la declaración de un control *Button*. Lo interesante de este control es que genera un evento cuando se hace click sobre él que llama a la función *btnIzq2_Click* definida en la parte de programación dinámica.

6.2 Comunicación con la base de datos

La comunicación con la base de datos se realiza desde el código estático. Como la aplicación se basa en la visualización de datos, las consultas enviadas a la base de datos son sólo de lectura de datos, por lo que sólo se utilizarán instrucciones *SELECT*.

Las consultas a realizar se definen mediante controles *SqlDataSource*. Aunque estos controles sean los enlaces con la base de datos, ellos no extraen los datos por sí solos. Para ello han de ser asociados a otros controles capaces de contener datos, tales como gráficas o controles *GridView*.

El programa de visualización Web realiza dos consultas a la base de datos. La primera consulta extrae de la tabla VARIABLE la columna de una estación e introduce los datos en un control *GridView* **(Figura 6-2)**.

```
<asp:GridView ID="gv" runat="server" AutoGenerateColumns="False" DataSourceID="SqlDataSource1">
    <Columns>
        <asp:CheckBoxField DataField="S401" HeaderText="S401" SortExpression="S401" />
    </Columns>
</asp:GridView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="Data Source=SAICA-SERVER;Initial Catalog=SAICA_DB;Integrated Security=True"
    SelectCommand="SELECT [S401] FROM [VARIABLES]">
</asp:SqlDataSource>
```

Figura 6-2. Consulta a la tabla VARIABLE

La segunda consulta se realiza para leer los datos de la tabla de la estación que se va a mostrar en la gráfica **(Figura 6-3)**. El control *SqlDataSource* es asociado a la gráfica de datos para la representación.

La función *CONVERT* dentro de la sentencia SQL transforma el dato fecha hora a una cadena de caracteres con el mismo formato de fecha que el que se utiliza en los controles *tbIni* y *tbFin*.

```
<asp:SqlDataSource ID="SqlDataSource" runat="server"
    ConnectionString="Data Source=SAICA-SERVER;Initial Catalog=SAICA_DB;Integrated Security=True"
    SelectCommand="SELECT * FROM [S401]"
    FilterExpression="CONVERT(smalldatetime, CONVERT(char(19), Fecha_Hora, 113), 113) >= '{0}' AND CONVERT(smalldatetime, CONVERT(char(19), Fecha_Hora, 113), 113) <= '{1}'"
    ProviderName="System.Data.SqlClient" CacheDuration="300" EnableCaching="True">
    <FilterParameters>
        <asp:ControlParameter ControlID="tbIni" PropertyName="Text"/>
        <asp:ControlParameter ControlID="tbFin" PropertyName="Text"/>
    </FilterParameters>
</asp:SqlDataSource>
```

Figura 6-3. Consulta a la tabla S401

6.3 Interfaz de usuario

El usuario que desea consultar los datos de una estación de una de las redes de seguimiento del proyecto, debe abrir un navegador Web en su equipo e introducir la dirección del servidor de la red donde se encuentra esa estación.

Al introducir la dirección del servidor, se carga en su navegador la página de selección de estación. Esta página **(Figura 6-4)** muestra el mapa de la cuenca con los puntos donde se encuentran las estaciones. Al hacer click sobre alguno de estos puntos, se abre en una pestaña nueva la página con los datos de la estación.

Figura 6-4. Página de selección de estación

La página de estación **(Figura 6-5)** es la página que muestra los valores de las medidas de la estación. Los datos mostrados tienen ciertas restricciones exigidas por el cliente:

- Sólo se muestran datos válidos.
- Los datos que se representan pertenecen a la semana anterior a la semana en la que se hace la consulta.
- Las variables que se pueden visualizar en la aplicación Web son las designadas por el técnico del centro de control mediante las variables mostradas por defecto en el programa de validación.

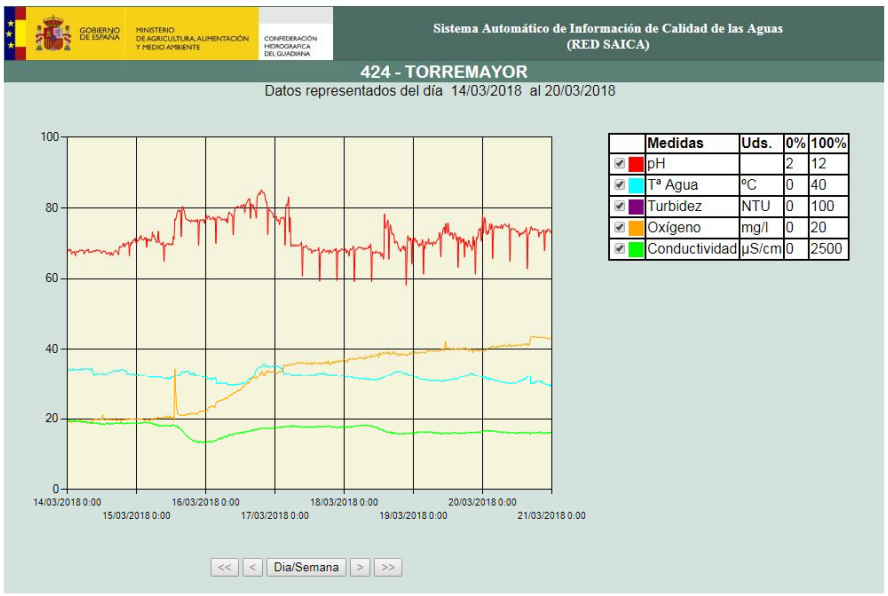

Figura 6-5. Página de la estación

En la parte superior de la pantalla se muestra el membrete de Confederación Hidrográfica del Guadiana y la red de seguimiento consultada. Bajo esto aparece el código y el nombre de la estación. Debajo del nombre de la estación se informa del periodo de datos que se puede visualizar en ese momento. Los datos son representados en la gráfica. Los valores de los datos vienen mostrados en porcentajes. A la izquierda de la gráfica se encuentra una tabla con las variables que pueden mostrarse. En esta tabla, el usuario puede deshabilitar la representación de alguna de estas variables y conocer las unidades en las que están medidas y el máximo y mínimo de los valores mostrados en la gráfica. El control de la línea temporal se sitúa justo debajo de la gráfica. Éste permite mostrar todos los datos de la semana o mostrarlo por días, además de desplazarte por el eje de abscisas mediante saltos de 1 día o de 12 horas cuando se están representando por días.

El diseño estático de la pantalla de selección de estación consiste en un control *ImageMap* **(Figura 6-6)**. Este control muestra una imagen sobre la que se mapean zonas activas que generan llamadas al servidor o navegan a una dirección especificada. La página de selección no tiene código dinámico.

```
<asp:ImageMap ID="ImageMap1" ImageUrl="~/saica/image/MapaSaica_v2.jpg" runat="server">
    <asp:CircleHotSpot Radius="6" X="358" Y="214"
        HotSpotMode="Navigate"
        NavigateUrl="~/saica/401"
        target="~/saica/401.aspx"
        AlternateText="401 - ENTRERRIOS"
        />
    <asp:CircleHotSpot Radius="6" X="320" Y="215"
        HotSpotMode="Navigate"
        NavigateUrl="~/saica/402"
        target="~/402.aspx"
        AlternateText="402 - MEDELLÍN"
        />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot ... />
    <asp:CircleHotSpot Radius="6" X="60" Y="453"
        HotSpotMode="Navigate"
        NavigateUrl="~/saica/430"
        target="~/430.aspx"
        AlternateText="430 - BOCACHANZA"
        />
</asp:ImageMap>
```

Figura 6-6. Control *ImageMap*. Selección de estación

Como se ve en la imagen anterior, cada estación tiene su propia página. Las páginas son exactamente iguales, lo único que se modifica es el código y el nombre de la estación y las consultas realizadas a la base de datos. En cada página de estación, las consultas se realizan a los datos de esa estación.

El diseño estático de una página de estación define la apariencia de la página y las conexiones con la base de datos. En primer lugar, utiliza dos controles *Label* para mostrar la fecha de inicio y la fecha final de la semana representada **(Figura 6-7)**.

```
<div class="contenty2" style="text-align:center; font-family:Arial">
    Datos representados del día
     <asp:Label ID="lblFechaIni" runat="server"></asp:Label>
     al
    <asp:Label ID="lblFechaFin" runat="server"></asp:Label>
     

</div>
```

Figura 6-7. Controles *lblFechaIni* y *lblFechaFin*

El siguiente control que se define es la gráfica **(Figura 6-8)**. La gráfica está asociada a la fuente de datos *SqlDataSource* **(Figura 6-3)**. De esta forma la gráfica consigue los datos de la estación de medidas. Dentro de la definición de la gráfica, se definen los colores que van a tomar las variables, las series que van a contener los datos de las variables y el área de representación donde el eje X representa el tiempo y el eje Y el porcentaje.

Las series se definen para las diez variables que se pueden representar en todas las estaciones, aunque no siempre estén habilitadas. Los campos *Id* de estas variables son *1*, *2*, *3*, *4*, *5*, *7*, *8*, *9*, *12* y *14*.

```
<asp:Chart ID="Chart1" runat="server" Width="680px" Height="480px" DataSourceID="SqlDataSource" Palette="None"
    PaletteCustomColors="Red; Cyan; Purple; Orange; Lime; DarkGreen; HotPink; Blue; OrangeRed; DodgerBlue; Transparent"
    BackColor="212,226,222" OnCustomize="Chart1_Customize">
    <series>
        <asp:Series Name="Series1" ChartType="Spline" XValueMember="Fecha_Hora"
            YValueMembers="Porcentaje, Validez, Variable" XValueType="DateTime" YValuesPerPoint="3">
        </asp:Series>
        <asp:Series Name="Series2" ChartType="Spline" XValueMember="Fecha_Hora"
            YValueMembers="Porcentaje, Validez, Variable" XValueType="DateTime" YValuesPerPoint="3">
        </asp:Series>
        <asp:Series ...>...</asp:Series>
        <asp:Series ...>...</asp:Series>
        <asp:Series ...>...</asp:Series>
        <asp:Series ...>...</asp:Series>
        <asp:Series ...>...</asp:Series>
        <asp:Series ...>...</asp:Series>
        <asp:Series ...>...</asp:Series>
        <asp:Series Name="Series14" ChartType="Spline" XValueMember="Fecha_Hora"
            YValueMembers="Porcentaje, Validez, Variable" XValueType="DateTime" YValuesPerPoint="3">
        </asp:Series>
    </series>
    <chartareas>
        <asp:ChartArea Name="ChartArea1" BackColor="Beige">
            <AxisX>
                <LabelStyle Format="g" />
            </AxisX>
            <AxisY Maximum ="100" Minimum ="0"></AxisY>
            <AxisY2 Maximum ="100" Minimum ="0"></AxisY2>
        </asp:ChartArea>
    </chartareas>
</asp:Chart>
```

Figura 6-8. Control *Chart*. Página de estación

Los botones del control de la línea temporal **(Figura 6-9)**, generan un evento cuando se hace click sobre ellos.

```
<p style="background-color:rgb(212,226,222)">
    <asp:Button id="btnIzq2" Text="<<" OnClick="btnIzq2_Click" runat="server" />
    <asp:Button id="btnIzq" Text="<" OnClick="btnIzq_Click" runat="server"/>
    <asp:Button id="btnDiaSem" Text="Dia/Semana" OnClick="btnDiaSem_Click" runat="server"/>
    <asp:Button id="btnDer" Text=">" OnClick="btnDer_Click" runat="server"/>
    <asp:Button id="btnDer2" Text=">>" OnClick="btnDer2_Click" runat="server"/>
</p>
```

Figura 6-9. Controles del eje X

Por último, se define una tabla a la derecha de la gráfica donde se eligen las variables que se representan en la gráfica. Por cada variable, la tabla contiene: un control *CheckBox* para seleccionar si la variable es dibujada, el color con el que se dibuja la variable, el nombre de la variable, la unidad de la medida, el valor mínimo de la representación y el valor máximo.

En el código dinámico se definen los procesos a ejecutar cuando sucede un evento en la página. Cada vez que un evento se activa, la página mostrada en el explorador Web del usuario insta al servidor a ejecutar la función a la que llama dicho evento. El servidor ejecuta la petición y vuelve a enviar al usuario la página con los cambios efectuados. Mientras el servidor está creando la nueva pantalla pueden suceder nuevos eventos. El servidor los atiende antes de enviar la página.

Uno de esos eventos que sucede mientras el servidor está creando la página, es el evento generado por la gráfica cuando empieza a ser dibujada. Este evento llama a la función *Chart1_Customize*.

La función *Chart1_Customize* **(Figura 6-10)** filtra los datos de cada serie por variable y luego los vuelve a filtrar para que sólo aparezcan en la gráfica los datos válidos.

```
protected void Chart1_Customize(object sender, EventArgs e)
{
    int i;
    for (i = 1; i < 11; i++)
    {
        Chart1.DataManipulator.FilterMatchedPoints = false;
        if(i < 9)
            Chart1.DataManipulator.Filter(CompareMethod.EqualTo ,
                Convert.ToInt32(Chart1.Series[i - 1].Name.Substring(6,1)),
                Chart1.Series[i - 1], Chart1.Series[i - 1], "Y3");
        else
            Chart1.DataManipulator.Filter(CompareMethod.EqualTo,
                Convert.ToInt32(Chart1.Series[i - 1].Name.Substring(6, 2)),
                Chart1.Series[i - 1], Chart1.Series[i - 1], "Y3");

        Chart1.DataManipulator.Filter(CompareMethod.EqualTo, 1,
            Chart1.Series[i - 1], Chart1.Series[i - 1], "Y2");
    }
}
```

Figura 6-10. Función *Chart1_Customize*

Los demás controles generan un evento que cambia una propiedad o una variable interna y vuelve a repintar la gráfica. Un ejemplo es el botón que avanza medio día en la gráfica. Este botón llama a la función *btnDer_Click* **(Figura 6-11)**. Esta función toma la fecha final que se estaba representando hasta ese momento y le suma 12 horas. Comprueba que esa fecha no se salga de la restricción de fechas impuesta. En el caso de incumplir la restricción, ajusta la fecha final a la última fecha que se puede representar. Una vez se tiene la fecha final, calcula la fecha inicial restándole 1 día a la fecha final. Cuando tiene ambos valores, configura el área de representación para que se vuelva a pintar la gráfica.

```
protected void btnDer_Click(object sender, EventArgs e)
{
    dateFin = Convert.ToDateTime(tbFin.Text).Add(TimeSpan.FromHours(12));
    if (dateFin >= Convert.ToDateTime(tbFinSem.Text))
    {
        dateFin = Convert.ToDateTime(tbFinSem.Text);
        btnIzq2.Enabled = true;
        btnIzq.Enabled = true;
        btnDer.Enabled = false;
        btnDer2.Enabled = false;
    }
    dateIni = dateFin.Subtract(TimeSpan.FromDays(1));

    tbFin.Text = dateFin.ToString();
    tbIni.Text = dateIni.ToString();

    Chart1.ChartAreas["ChartArea1"].AxisX.Maximum = dateFin.ToOADate();
    Chart1.ChartAreas["ChartArea1"].AxisX.Minimum = dateIni.ToOADate();
}
```

Figura 6-11. Función *btnDer_Click*

El resto de eventos generados por la página son similares a éste. Por este motivo, explicando este proceso se da por explicado el resto de procesos.

7 FORMULARIO ACCESS

Los formularios creados en Access son la herramienta ofrecida al técnico del centro de control para la exportación de datos y la generación de informes con los datos de las variables almacenados en la base de datos. Los formularios generados para el servidor SAICA y para el servidor REACAR son similares. La única diferencia existente es el servidor SQL al que se realizan las consultas y el nombre de las tablas de estación de las que se extraen los datos.

En este caso, se va a mostrar el formulario de consulta para la red REACAR.

En primer lugar, es necesario crear un vínculo con la base de datos SQL. Para ello se utiliza el asistente *Obtener datos externos: Base de datos ODBC*. Se selecciona la opción *Vincular al origen de datos creando una tabla vinculada*. A continuación se pulsa sobre el botón *Nuevo...* y se selecciona de la lista la opción *SQL Server*. Se le da un nombre al archivo donde se va a guardar la conexión. Después de darle nombre al archivo, se introduce el servidor SQL. En el siguiente formulario, se selecciona el tipo de autentificación de inicio de sesión, en este caso, autenticación de Windows. Se selecciona la base de datos a la que se quiere acceder y se seleccionan las tablas a vincular.

Estas tablas se muestran en la barra de objetos de la izquierda. Son vínculos a la base de datos, por lo que son peligrosas ya que se podrían modificar los datos de la base de datos a través de ellas. Por ese motivo se ocultan para que no se muestren en la barra de objetos.

A continuación, se crean las consultas que generarán las tablas a las que tendrá acceso el usuario. Se crea una consulta por cada tabla de estación. Las consultas se diseñan directamente en lenguaje SQL puesto que es la única forma de crear una tabla con los datos que sea entendible por el usuario.

La consulta a la estación R16 se muestra en la siguiente figura **(Figura 7-1)**. La consulta consiste en una instrucción *SELECT* que se estructura de la siguiente forma:

- Se seleccionan las columnas que se van a mostrar y se le da un nuevo nombre a la columna de la tabla que se va a crear mediante la cláusula *AS*. Por ejemplo, se selecciona la columna *Valor* de la tabla *A* y se pasa a llamar *pH*.
- Mediante la cláusula *INTO* se selecciona la tabla donde se va a introducir los datos. Si no existe se crea, y si existe se sobreescriben sus datos.

- Se indican las tablas de donde se leen los datos. En esta consulta todos los datos provienen de la misma tabla, pero debido a que se van a ordenar en distintas columnas dependiendo de la variable, se las renombra mediante la cláusula *AS* para que se traten como tablas diferentes.
- A partir de la cláusula *WHERE* se introducen las restricciones necesarias para que la tabla muestre los datos seleccionados de manera entendible para el usuario.
    - Las fechas tienen que ser las mismas para los datos de la misma fila.
    - Cada tabla corresponde a una variable distinta.
    - Las fechas que se rescatan son las comprendidas entre las fechas que se seleccionen en el formulario

Consulta_R16

```
SELECT A.Fecha_Hora AS Fecha_Hora, A.Valor AS pH, A.Validez AS [pH Validez],
  B.Valor AS [Tª Agua], B.Validez AS [Tª Agua Validez],
  C.Valor AS Turbidez, C.Validez AS [Turbidez Validez],
  D.Valor AS Oxígeno, D.Validez AS [Oxígeno Validez],
  E.Valor AS Conductividad, E.Validez AS [Conductividad Validez],
  F.Valor AS RedOx, F.Validez AS [RedOx Validez],
  G.Valor AS SAC, G.Validez AS [SAC Validez]
INTO R16
FROM dbo_R16 AS A,
  dbo_R16 AS B,
  dbo_R16 AS C,
  dbo_R16 AS D,
  dbo_R16 AS E,
  dbo_R16 AS F,
  dbo_R16 AS G
WHERE A.Fecha_Hora = B.Fecha_Hora
  AND A.Fecha_Hora = C.Fecha_Hora
  AND A.Fecha_Hora = D.Fecha_Hora
  AND A.Fecha_Hora = E.Fecha_Hora
  AND A.Fecha_Hora = F.Fecha_Hora
  AND A.Fecha_Hora = G.Fecha_Hora
  AND A.Variable = 1
  AND B.Variable = 2
  AND C.Variable = 3
  AND D.Variable = 4
  AND E.Variable = 5
  AND F.Variable = 6
  AND G.Variable = 7
  AND A.Fecha_Hora BETWEEN [Formularios]![Consulta]![txtDTini] And [Formularios]![Consulta]![txtDTfin];
```

Figura 7-1. Consulta SQL en Access

El formulario creado **(Figura 7-2)** consiste en un control *ComboBox* para la selección de la estación, dos controles para la selección de la fecha inicial y la fecha final de la consulta y un botón para proceder con ella.

Figura 7-2. Formulario Access

El control *ComboBox* muestra las estaciones dado a la consulta introducida en la propiedad *Origen de fila* **(Figura 7-3)**.

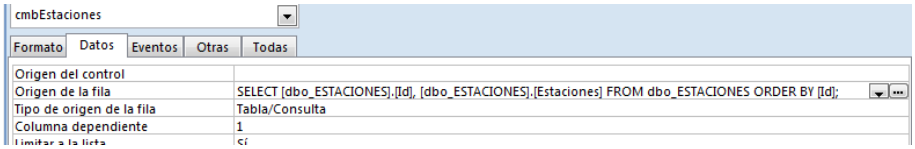

Figura 7-3. Configuración *ComboBox* formulario Access

Los controles de fechas llaman a un formulario **(Figura 7-4)** donde elegir las fechas. Este formulario está formado por un campo donde introducir la fecha que al ser pulsado abre un calendario, un campo donde introducir la hora, un campo para los minutos y dos botones para aceptar o cancelar la selección.

Figura 7-4. Formulario *Introducir fecha y hora*

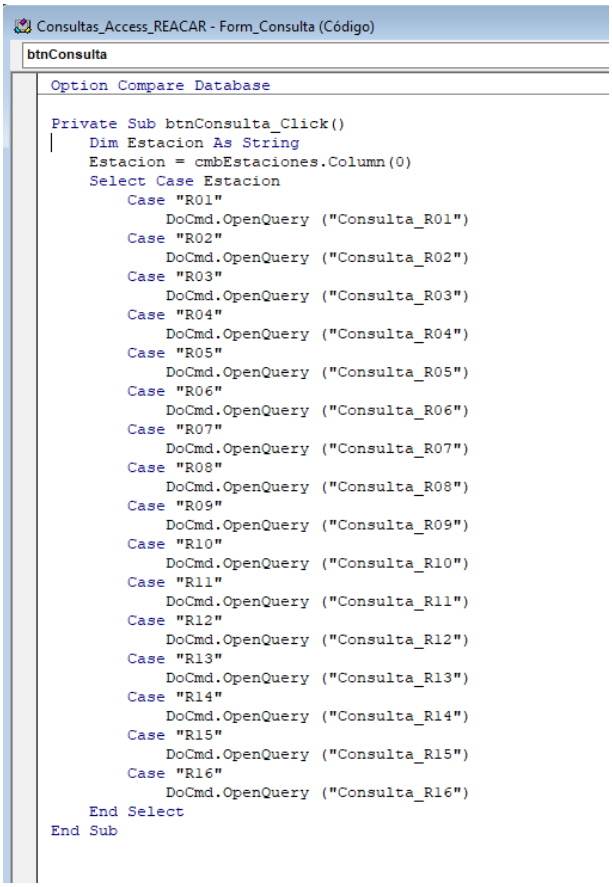

Figura 7-5. Función *btnConsulta_Click*

El botón *Generar Consulta* llama a una función creada en el editor de código de Visual Basic que ofrece el paquete Microsoft Office **(Figura 7-5)**. Esta consulta ejecuta la consulta de creación de tabla que pertenece a la estación seleccionada en el *ComboBox*.

De esta manera, se genera la tabla con los datos seleccionados **(Figura 7-6)**.

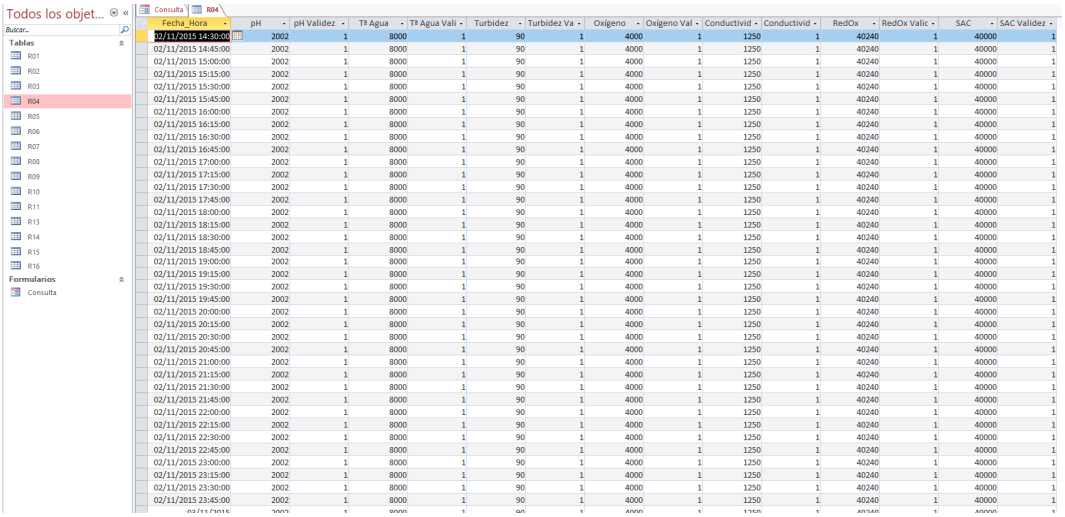

Figura 7-6. Tabla *R04* en Access

Una vez generadas las tablas que el técnico necesite, éste realiza los informes y las exportaciones de datos con las herramientas que ofrece Access.

8 CONCLUSIONES Y LÍNEAS FUTURAS

Tras la finalización del proyecto y pasado un tiempo desde su entrega, el resultado final ha sido satisfactorio. Los objetivos marcados al inicio han sido completados, tanto los objetivos impuestos por el cliente, como los objetivos personales y por parte de la empresa.

En primer lugar, se ha logrado complacer al cliente. El resultado que ellos esperaban era algo igual a lo que ya tenían, pero introduciendo los cambios que pedían. Lo que se les ha entregado cumple con esas exigencias y, además, se les ha aportado ciertas funcionalidades que no poseían, resultando ser un sistema más sencillo para el técnico y dándole la posibilidad de gestionar todo el sistema. Con ello, se han superado las expectativas iniciales del cliente.

En el ámbito personal, el aprendizaje ha sido amplio. Ha servido para introducirme en el mundo de los sistemas con bases de datos, programar en un lenguaje nuevo y saber establecer una relación profesional con el cliente para poder contentar a todas las partes implicadas.

La empresa ha adquirido nuevas tecnologías para el desarrollo, nuevo know-how y una relación más estrecha con el cliente. El coste que ha tenido el desarrollo de este proyecto ha demostrado ser una inversión a futuro inmejorable. Todo el conocimiento y las herramientas logrados aquí han sido utilizados en posteriores trabajos.

Después de analizar los resultados y el proceso para alcanzarlos, se ha llegado a las siguientes conclusiones:

- Es imprescindible colaborar con el cliente en la fase de desarrollo y ejecución del proyecto para maximizar el éxito del trabajo. Muchas veces, los clientes están poco interesados en el desarrollo, en cómo está hecho y qué tecnologías utiliza. El interés del cliente radica en su funcionamiento, que sea útil, cumpla sus expectativas y no tenga un uso complejo. Hay que conseguir implicar al cliente durante el desarrollo. Con ello se consigue que éste comprenda mejor las posibilidades existentes, encuentre nuevas necesidades que no conocía que tenía y entienda mucho mejor el funcionamiento de la solución que se le va a entregar.
- La aportación de valores añadidos intangibles ayudan a conseguir la satisfacción del cliente y fortalecer nuestra marca, nuestro valor en el mercado y destacarnos de la competencia. Estos intangibles son la cercanía con el cliente, la personalización de los productos entregados o la atención post-ejecución dada a los usuarios que utilicen lo que se ha desarrollado.
- Cumplir las demandas del cliente no es suficiente. Hay que intentar conocer las necesidades reales del cliente y superar sus expectativas. Conseguir esto es conseguir su confianza y lealtad, generando el interés por parte de ellos de colaborar con nosotros en futuros trabajos.
- El éxito del proyecto se garantiza a través de la buena gestión de los recursos y de la planificación. Especialmente en este proyecto, donde no poseía conocimientos suficientes sobre bases de datos y lenguaje C#, ha sido muy importante planificar correctamente el desarrollo, calculando tiempos de aprendizaje, pruebas y ejecución de las distintas partes en las que se ha dividido el plan de trabajo.

- Las soluciones aportadas al cliente deben servir como base para futuros trabajos. Las soluciones deben poderse adaptar a otros proyectos y formar parte del know-how de la empresa. Esto facilitará el desarrollo de futuros trabajos, abaratando costes y reduciendo tiempos.
- El aprendizaje a través del trabajo real es la mejor forma de adquirir y reforzar conocimientos. Los casos prácticos esclarecen conceptos teóricos y ayudan a su comprensión y asimilación. A caminar se aprende andando.
- Hay que tener una actitud abierta a detectar errores. No se puede evitar cometer errores. Su detección y posterior análisis permite ir mejorando tanto técnicamente como profesionalmente, especialmente en trabajos innovadores. Una vez más, se ha de implicar a los clientes en la detección de errores, planificando fases de pruebas para ellos y fases de análisis por parte de la empresa. A través del análisis de errores se pueden descubrir otros errores ocultos que podrían haber aparecido en situaciones críticas.
- Menos es más. A la hora de desarrollar un código, hay que tratar que sea lo más sencillo posible, buscando formas de reducir su tamaño y separando procesos diferentes en funciones distintas. Esto ayuda a la depuración y comprensión del código. Además, si se pretende reutilizar parte de ese código en un futuro, ya sea por uno mismo o por un compañero, es más fácil cuando está claro y limpio.

Las posibles ampliaciones de este proyecto vienen dadas principalmente por el crecimiento de las redes de seguimiento y por la introducción de nuevas variables a medir. El sistema ya incluye nuevas variables que no están siendo tomadas aún. Aparte, se ha reservado espacio en los paquetes de datos de los PLC para añadir algunas más. En el caso de añadir aún más variables, habría que modificar un poco el programa del PLC y la función que lee los paquetes de datos de la aplicación de validación y el formulario donde se representan. La base de datos está preparada para añadir tantas variables como se precise.

En el caso de añadir nuevas estaciones, sólo habría que modificar la base de datos, añadiendo una nueva tabla por cada estación para los datos de esta y sumando un nuevo campo por estación en la tabla VARIABLES para las variables representadas por defecto.

Otras ampliaciones podrían ser añadirle funcionalidades al programa principal. Una de esas funcionalidades podría ser la creación de informes directamente desde este programa. Otra posibilidad es permitir ampliar o reducir zoom en la gráfica para observar los datos cuando se escapen de los rangos de valores mostrados.

Se podría configurar el servidor como un servidor de aplicaciones para que el técnico pueda realizar el control de las redes desde fuera del centro de control. Esto deslocalizaría el lugar de trabajo y permitiría a otros responsables acceder al sistema desde sus equipos.

Gracias al conocimiento adquirido en este proyecto, se han desarrollado otros trabajos y he ido incrementando mis habilidades con estas tecnologías. Un ejemplo de ello es que actualmente me encuentro desarrollando un sistema MES (Manufacturing Execution System) de una fábrica con varias líneas de producción relacionando el control de la fábrica con los procesos de planificación de la empresa.

Para finalizar esta memoria, me gustaría agradecer a mi tutora María Teresa Ariza Gómez, la guía y ayuda prestada durante el desarrollo de esta memoria. Le doy las gracias por su tiempo y comprensión además de por sus consejos y correcciones.

ANEXOS

Los anexos que se incluyen en este documento tienen el objetivo de complementar la información dada durante la memoria. Esta información no ha sido incluida dentro de la memoria al considerarse que carecen de peso en el desarrollo del proyecto.

ANEXO I

Configuración SQL Server

El gestor de bases de datos SQL Server puede ser configurado para realizar tareas de mantenimiento de las distintas bases de datos que administra. En este anexo se explica el procedimiento de configuración para que se lleven a cabo estas tareas.

Al instalar SQL Server no sólo se instala el motor o servidor de bases de datos, también se instalan otras instancias para otras funciones, ya puedan ser la generación de informe o la monitorización de los procesos del motor.

Para realizar planes de mantenimiento es necesario el Agente SQL Server. Para activarlo es necesario abrir *Sql Server Configuration Manager* **(Figura I-1)**. En la ventana que aparece se selecciona *Servicios de SQL Server* para que muestre los servicios disponibles. En los servicios mostrados se selecciona *Agente SQL Server* y se inicia su ejecución. También es necesario poner el modo de inicio en automática para que este servicio esté siempre ejecutando.

Una vez activado el Agente SQL Server, se procede a abrir el administrador de SQL Server, *Microsoft SQL Server Management Studio*. En el explorador de objetos se abre el servidor local, se abre la carpeta *Administración* y dentro de esta carpeta se selecciona *Planes de mantenimiento* con el click derecho del ratón y se elige *Nuevo plan de mantenimiento…* .

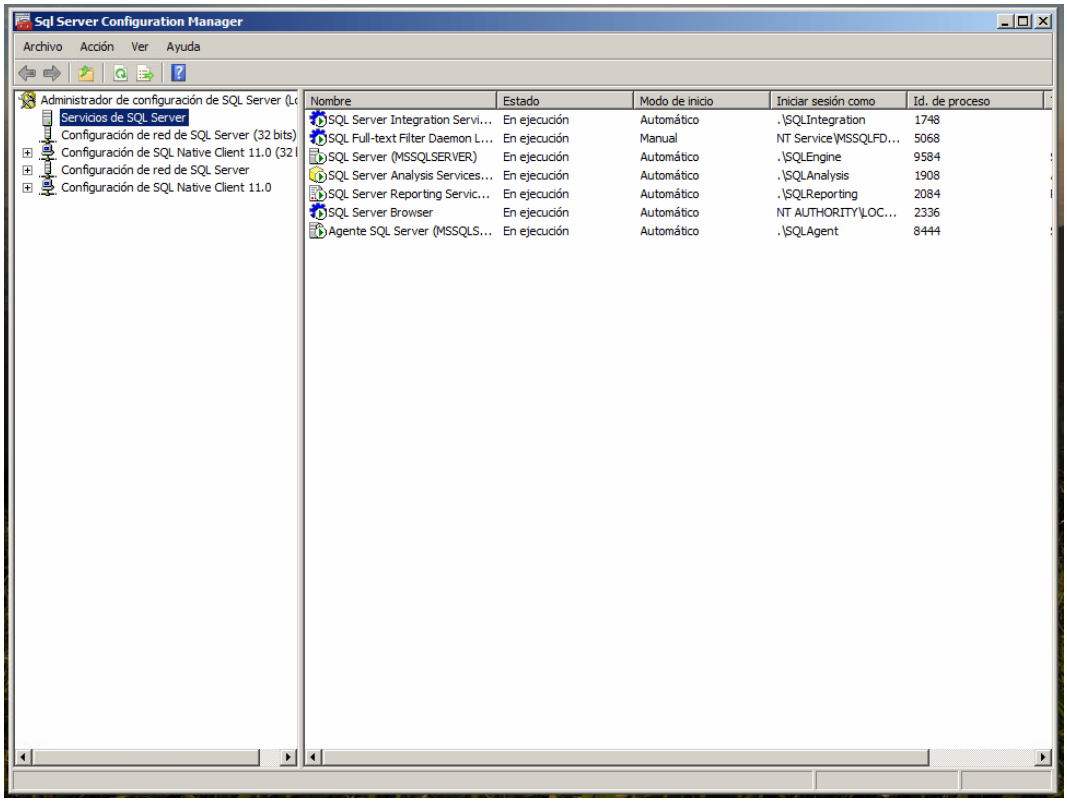

Figura I-1. Sql Server Configuration Manager

Se escribe el nombre del nuevo plan y se agregan subplanes. Los subplanes que se han generado para el mantenimiento de las bases de datos son:

- Creación de una copia de seguridad completa **(Figura I-2)**. Este subplán se ejecuta semanalmente.

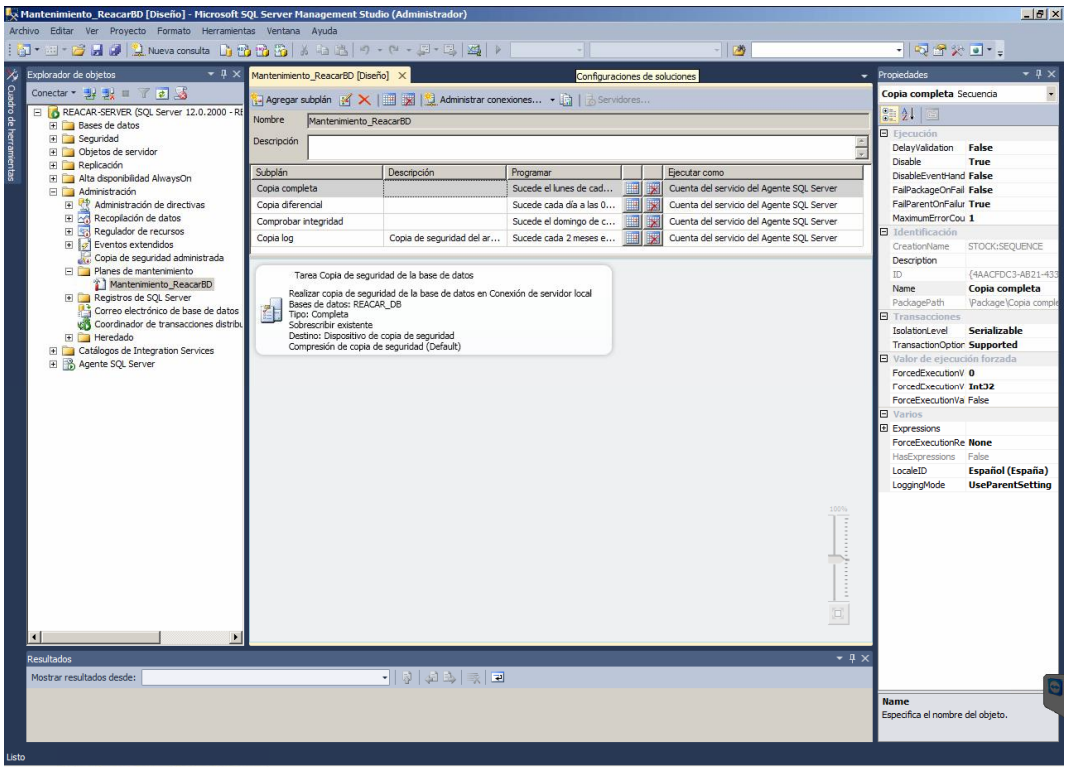

Figura I-2. Planes de mantenimiento y tarea de copia de seguridad completa

- Creación de una copia de seguridad diferencial **(Figura I-3)**. Una copia diferencial consiste en guardar sólo los datos que difieren de la copia de seguridad completa. Se realiza a diario para que no se pierdan los cambios efectuados en la base de datos durante el día.

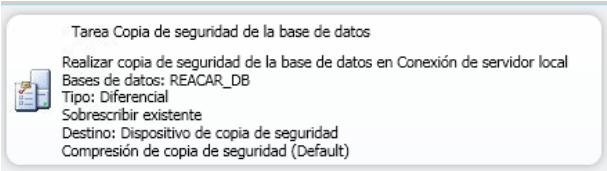

Figura I-3. Tarea de copia de seguridad diferencial

- Comprobación de la integridad **(Figura I-4)**. Esta tarea revisa que no existan datos corruptos dentro de la base de datos. Esta tarea se efectúa mensualmente.

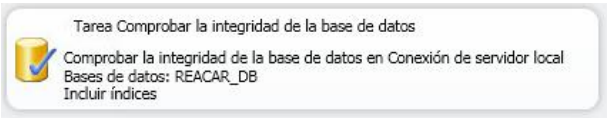

Figura I-4. Tarea de comprobación de integridad

- Copia de seguridad del archivo de registros **(Figura I-5)**. El archivo de registro de transacciones es un archivo donde se almacenan todos los cambios efectuados en la base de datos para poder deshacer los cambios uno a uno. Este archivo está configurado de manera que crezca hasta un tamaño máximo para que no deje sin espacio en disco al sistema para almacenar los datos verdaderamente importantes, los datos de las estaciones de medidas. El problema de este archivo es que se llena con rapidez, y para poder vaciarlo es necesario realizar una copia completa de este archivo. La copia de seguridad del registro de transacciones se realiza una vez cada dos meses.

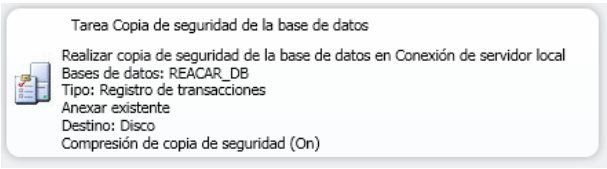

Figura I-5. Tarea de copia de seguridad del registro de transacciones

ANEXO II

Tareas de copias de seguridad

Siempre es conveniente poseer una estrategia para cuando ocurren errores. En primer lugar, se ha intentado minimizar esos errores obteniendo un hardware de calidad y redundante. El servidor que se ha instalado posee una configuración RAID 0+1 para sus discos duros. Esta configuración crea una copia exacta (espejo) de la mitad de los discos duros del sistema en la otra mitad. Además, los discos duros que poseen los servidores son de intercambio en caliente (Hot-Swap) que permite la sustitución de algún disco en mal estado por otro nuevo de sus mismas características sin tener que parar el sistema.

A pesar de ello, puede producirse cualquier otro tipo de avería que impida el funcionamiento de algún servidor. Para ello se ha programado una tarea para que se realice una copia del último archivo de copia de seguridad de la base de datos en el servidor perteneciente a la otra red de seguimiento. Es decir, el servidor de SAICA poseerá una copia de seguridad de la base de datos REACAR, y el servidor REACAR de la base de datos SAICA. Además de la copia de seguridad, se ha dejado para su instalación en cada servidor los programas de validación de los otros equipos adaptados para ejecutarse en el otro sistema. De esta forma, si hubiese que reparar el servidor de alguna de las dos redes, se podría seguir obteniendo datos de las estaciones de medidas sin más. Lo único que quedaría inactivo hasta la reparación del servidor sería la consulta Web de los datos.

- El primer paso para llevar a cabo este procedimiento es compartir la carpeta donde se almacenan los archivos de copias de seguridad.
- Después se concede acceso a esta carpeta al usuario que identifica al técnico de la sala de control mediante las opciones avanzadas de las propiedades de la carpeta.
- En el otro equipo se comprueba que se puede acceder a esa carpeta con la identificación de usuario y se crea una conexión a unidad de red dirigida a esa carpeta para que sea accesible por el sistema de forma permanente.
- Se crea un archivo .bat **(Figura II-1)** en el bloc de notas que copie la última copia de seguridad de la base de datos de la red de seguimiento del otro servidor en el disco duro del servidor que lo ejecute.

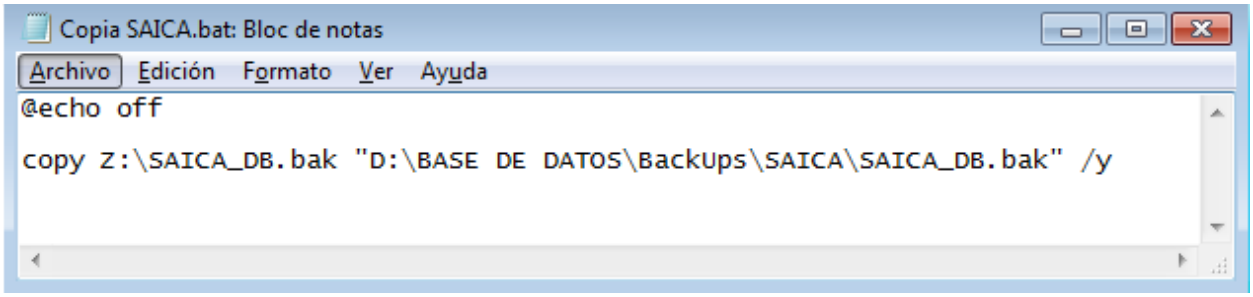

Figura II-1. Archivo.bat

- Se configura una tarea programada que ejecute el archivo .bat periódicamente. La copia se realizará semanalmente, un día después de que el servidor SQL haya generado la última copia de seguridad. Para ello se utiliza el programador de tareas de Windows Server.

ANEXO III

Configuración SYSMAC Gateway

Para que el programa de validación pueda llevar a cabo la comunicación con los distintos PLC que se encuentran en las estaciones de medidas, es necesario configurar el servidor. La aplicación que realiza la configuración de la red de comunicaciones requerida se llama SYSMAC Gateway. Este producto es un middleware desarrollado por la misma compañía que ha producido CX-Compolet, el método que se ha utilizado para interactuar con los PLC. SYSMAC Gateway aporta los sistemas de comunicación que CX-Compolet necesita actuando como driver de comunicaciones.

La configuración de la red se confecciona a través de la consola de SYSMAC Gateway **(Figura III-1)**. En ella se pueden configurar varios tipos de comunicaciones: comunicación directa a la red, tabla de etiquetas de registros de los PLC y memoria virtual de los PLC.

Puesto que el número de PLC al que hay que acceder es amplio y todas las direcciones IP no son conocidas en el momento del desarrollo, además del tipo de lectura de datos que se realiza en el programa de validación no va dirigido a varios registros sino que se recorre casi toda la memoria, se ha optado por la comunicación directa entre el programa y los PLC.

Para configurar esta red, se debe seleccionar *Communication Network* en el menú situado en la zona izquierda de la consola de SYSMAC Gateway.

Aparecerá en la consola esta sección con un apartado para configurar el servicio de comunicaciones y otro apartado donde crear y configurar los puertos de las redes.

En primer lugar, se ha de configurar el puerto de red que utilizará el complemento CX-Compolet del programa de validación. El puerto utilizado es el de la red Ethernet. Para configurarlo hay que seleccionarlo y pulsar el botón *Properties*. SYSMAC Gateway abre una ventana con las propiedades del puerto.

En esta ventana se selecciona la ID del puerto que va a utilizar CX-Compolet **(Figura 5-22)**, se elige la tarjeta de red del equipo por la que se realizará las comunicaciones (en caso de disponer de más de una) y se introduce la dirección IP del equipo donde se encuentra el programa de validación. Una vez configurado esto, se selecciona el *checkbox* para que el puerto se abra automáticamente cada vez que se inicie el servidor. Después se pulsa en *OK* para guardar la configuración.

Cuando ya se tiene configurado el puerto que se va a utilizar, se configura el servicio para que se inicie automáticamente eligiendo la opción *Auto* en el *combobox*. Inmediatamente, se pulsa sobre el botón *Start* para que comience el servicio.

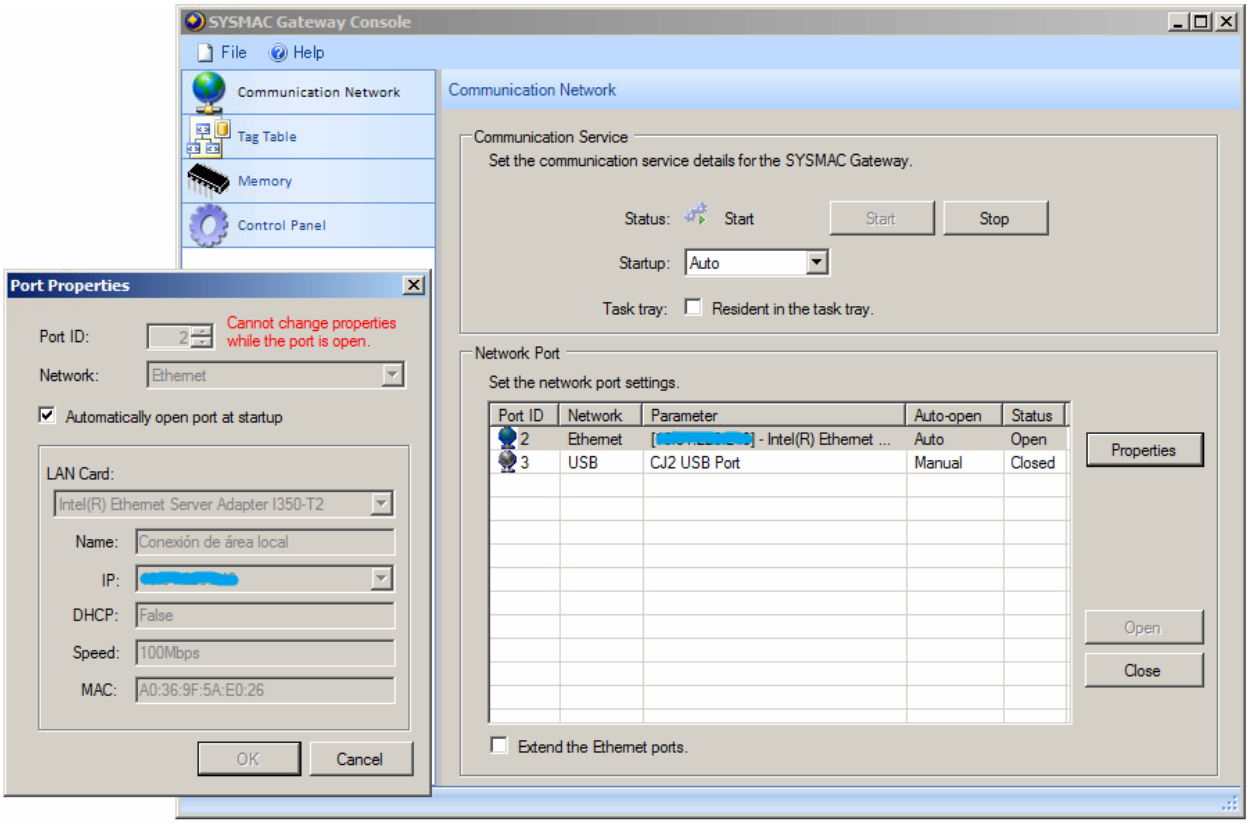

Figura III-1. Consola SYSMAC Gateway

ANEXO IV

Configuración de los servicios Web (IIS)

El servidor Web IIS es el servidor que se ha elegido para ofrecer la aplicación de visualización Web desarrollada en ASP.NET al exterior. En este anexo se explica cómo poner en marcha una aplicación Web posteriormente a haberla creado.

La instalación de IIS es tan sencilla como acceder al asistente para agregar roles y características y seleccionar el rol Servidor Web con las capacidades necesarias para la ejecución de un programa ASP.NET. En este rol se pueden activar muchas características, pero las capacidades imprescindibles en este caso son:

- ASP.NET y sus extensiones.
- La autenticación básica y de Windows.

Después de instalar IIS y de haber compilado y publicado el programa de visualización Web en un directorio dentro del servidor, se procede a configurar el sitio Web. Para ello se utiliza el Administrador de Internet Information Services (IIS) **(Figura IV-1)**.

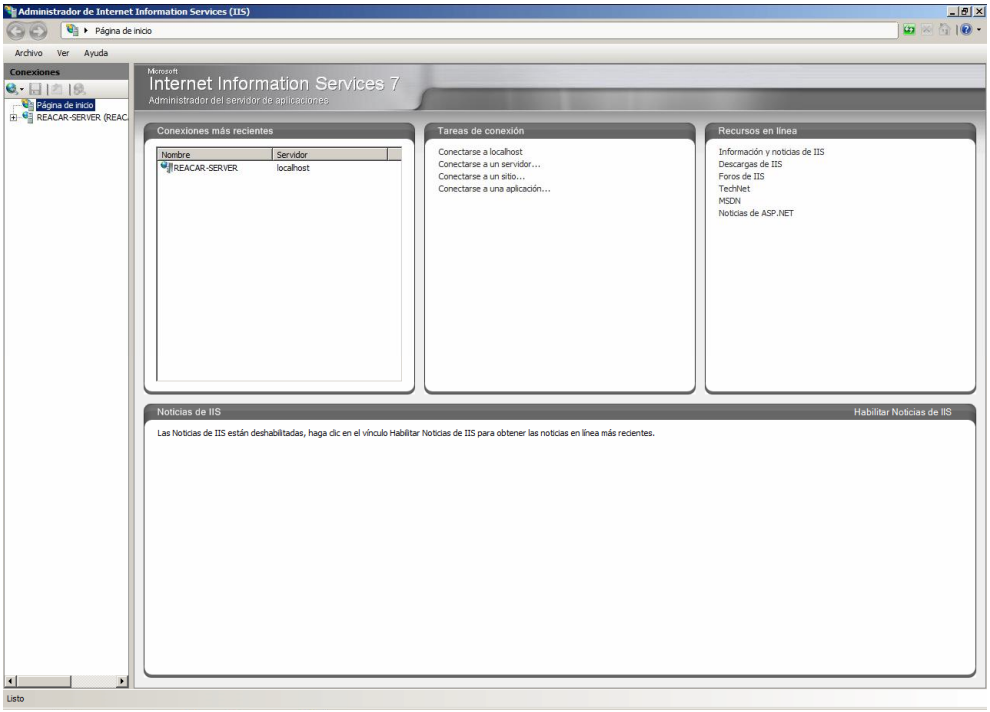

Figura IV-1. Administrador de IIS

En esta consola de administración aparece un sitio Web por defecto. Este sitio Web debe ser eliminado y a continuación se crea uno nuevo que se llamará REACARweb. Al sitio creado se le asigna una ruta de acceso física, el tipo de enlace *http* y el puerto por el que se accederá al servicio. Esta configuración se muestra a continuación **(Figura IV-2)**:

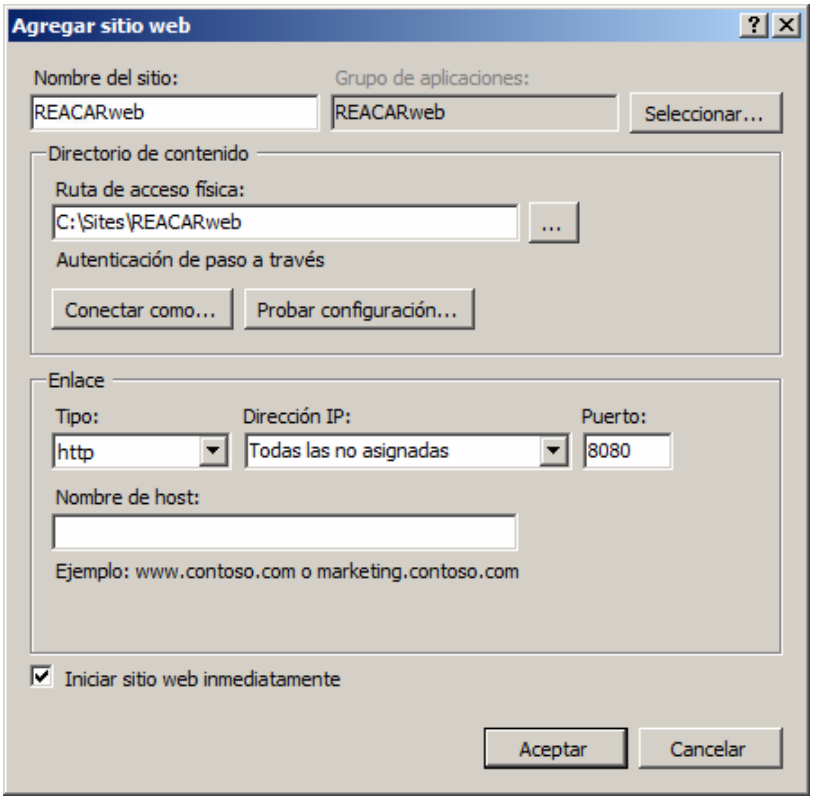

Figura IV-2. Configuración sitio Web

Una vez creado el sitio Web, se selecciona con el click derecho sobre él y se elige *Agregar aplicación*. Dentro del formulario que se abre, se introduce un alias para la aplicación dentro del grupo de aplicaciones y se elige la ruta donde se encuentra publicado físicamente el programa.

De esta forma, la aplicación ya es accesible desde la red interna de Confederación. Para hacerla accesible desde internet, el departamento de IT de Confederación deberá redireccionar a la dirección IP y al puerto TCP las peticiones que lleguen a la página Web.

BIBLIOGRAFÍA

[1] «Programas de seguimiento - Ministerio de agricultura, pesca, alimentación y medio ambiente» [En Línea]. Disponible: http://www.mapama.gob.es/es/agua/temas/estado-y-calidad-de-las-aguas/aguas-superficiales/programas-seguimiento/

[2] «Red de Estaciones automáticas - SAICA - Confederación Hidrográfica del Guadiana» [En Línea]. Disponible: http://www.chguadiana.es/cuenca-hidrografica/calidad-y-estado-de-las-masas-de-agua/aguas-superficiales/red-de-estaciones-automaticas-saica

[3] «CAS_RedSeguimiento.pdf - Ministerio de agricultura, pesca, alimentación y medio ambiente » [En Línea]. Disponible: http://sig.mapama.es/Docs/PDFServiciosProd2/CAS_RedSeguimiento.pdf

[4] «Review: Windows Server 2008 R2 ï V3 ï digital information news, analysis and insight» [En Línea]. Disponible: https://www.v3.co.uk/v3-uk/review/1954861/review-windows-server-2008-r2

[5] «New Features in Windows Server 2008 R2 ï IT Pro Today» [En Línea]. Disponible: http://www.itprotoday.com/management-mobility/new-features-windows-server-2008-r2

[6] «Documentación de SQL Server | Microsoft Docs» [En Línea]. Disponible: https://docs.microsoft.com/es-es/sql/sql-server/sql-server-technical-documentation

[7] «IDE de Visual Studio, editor de código, Team Services y Mobile Center» [En Línea]. Disponible: https://www.visualstudio.com/es/

[8] «Introduction to the C# Language and the .NET Framework | Microsoft Docs» [En Línea]. Disponible: https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

[9] «Overview of the .NET Framework | Microsoft Docs» [En Línea]. Disponible: https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview

[10] «Información general sobre ASP.NET | Microsoft Docs» [En Línea]. Disponible: https://docs.microsoft.com/es-es/aspnet/overview

[11] «CX-Compolet / SYSMAC Gateway | OMRON Industrial Automation» [En Línea]. No Disponible: http://www.ia.omron.com/data_pdf/cat/cx-compolet_v302-e1_10_1_csm1000258.pdf?=63

[12] «CX-Compolet / SYSMAC Gateway | OMRON España» [En Línea]. Disponible: https://industrial.omron.es/es/products/cx-compoletsysmac-gateway

[13] «Home: The Official Microsoft IIS Site» [En Línea]. Disponible: https://www.iis.net/

[14] «Introduction to IIS Architecture | Microsoft Docs» [En Línea]. Disponible: https://docs.microsoft.com/en-us/iis/get-started/introduction-to-iis/introduction-to-iis-architecture

[15] «Clase Chart (System.Windows.Forms.DataVisualization.Charting) - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/system.windows.forms.datavisualization.charting.chart(v=vs.110).aspx

[16] «Clase DataGridView (System.Windows.Forms) - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/system.windows.forms.datagridview(v=vs.110).aspx

[17] «Trabajar con conjuntos de datos en Visual Studio - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/8bw9ksd6.aspx

[18] «Create and configure datasets in Visual Studio - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/en-us/library/04y282hb.aspx

[19] «Clase SqlConnection (System.Data.SqlClient) - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/system.data.sqlclient.sqlconnection(v=vs.110).aspx

[20] «Cómo: Crear y ejecutar una instrucción SQL que devuelva filas ï MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/fksx3b4f.aspx

[21] «Clase SqlTransaction (System.Data.SqlClient) - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/system.data.sqlclient.sqltransaction(v=vs.110).aspx

[22] «Método SqlTransaction.Rollback (System.Data.SqlClient) - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/zayx5s0h(v=vs.110).aspx

[23] «Método SqlTransaction.BeginTransaction (System.Data.SqlClient) - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/86773566(v=vs.110).aspx

[24] «try-catch (Referencia de C#) | Microsoft Docs» [En Línea]. Disponible: https://docs.microsoft.com/es-es/dotnet/csharp/language-reference/keywords/try-catch

[25] «try-catch (C#) - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/0yd65esw(v=vs.80).aspx

[26] «Clase Chart (System.Web.UI.DataVisualization.Charting) - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/system.web.ui.datavisualization.charting.chart(v=vs.110).aspx

[27] «Seleccionar datos mediante el control SqlDataSource - MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/w1kdt8w2(v=vs.100).aspx

[28] «Clase SqlDataSource (System.Web.UI.WebControls)- MSDN - Microsoft» [En Línea]. Disponible: https://msdn.microsoft.com/es-es/library/w1kdt8w2(v=vs.100).aspx

[29] «Creación de una base de datos | Microsoft Docs» [En Línea]. Disponible: https://docs.microsoft.com/es-es/sql/relational-databases/databases/create-a-database?view=sql-server-2017

[30] «Programar tareas administrativas de SSAS con el Agente SQL Server | Microsoft Docs» [En Línea]. Disponible: https://docs.microsoft.com/es-es/sql/analysis-services/instances/schedule-ssas-administrative-tasks-with-sql-server-agent?view=sql-analysis-services-2017