Universidad Zaragoza
1542

Proyecto Fin de Carrera

Herramienta de diseño de sistemas basados en fuentes renovables para el sector agropecuario

Designing systems tool based on renewable sources for the agricultural sector

Autor

Jaume Montornés Cuartero

Director

José Luis Bernal Agustin

ESCUELA DE INGENIERIA Y ARQUITECTURA

2017

AGRADECIMIENTOS:

Mis agradecimientos a Virginia por su paciencia durante estos años, a Nayra por lo que sé que me quiso incondicionalmente (volveremos a vernos, pero aun no) y a Héctor y Amelia por darle sentido a cada día y darme fotos para cubrir la placa desmotivadora,
"Do it for them"

Herramienta de diseño de sistemas basados en fuentes renovables para el sector agropecuario

RESUMEN

En el presente proyecto se ha desarrollado un herramienta informática en entorno Windows destinado al cálculo de instalaciones energéticas basadas en fuentes renovables, encaminada tal aplicación a la automatización y simplificación del cálculo de tal tipo de instalaciones en entornos rurales y orientados al sector agropecuario, propiciando al usuario del mismo, un sistema generalista formado por una interface gráfica intuitiva y de fácil utilización.

En la actualidad las empresas del sector agropecuario, dada la tecnificación de las mismas demandan energía a fin de alimentar múltiples sistemas, generándose una aplicación informática versátil que pueda contemplar dichos elementos, sin presentarse restricciones de incorporación de puntos de consumos por parte del usuario. Así mismo se considera de utilidad la implantación de sistemas energéticos alternativos en empresas del sector agropecuario, dado que los consumos suelen ubicarse en zonas aisladas, lejos de núcleos urbanos o con dificultad de realizar transporte energético, siendo de muy fácil integración aplicaciones energéticas basadas en fuentes renovables.

La aplicación desarrollada en el presente proyecto se corresponderá a una interface gráfica para el cálculo de diferentes casuísticas de instalaciones renovables (fotovoltaicas aisladas de red, conectadas a red, etc…), utilizando para ello cuadros de dialogo intuitivos que guíen al usuario en el cálculo, gestionando el desarrollo conforme a pliegos técnicos específicos, de manera que corrijan, orienten y favorezcan la gestión del proyecto.

La aplicación de se ha programado en lenguaje C++ orientado a objetos, permitiendo esto un entorno modular, con menores errores y posibilidad de reutilización de código (objetos probados, robustos y fiables). Dicha programación orientada a objetos, propicia la reutilización de código en diferentes áreas del programa (aminorando la escritura de código), así como facilita su interpretación y comprensión, aligerando la labor de añadir, suprimir y/o modificar objetos, permitiendo esto la realización de modificaciones sencillas y eficientes.

Según esto el desarrollo se ha basado en cuadros de diálogo independientes, los cuales intercambian información y se conectarán entre ellos a través de las propias variables de cálculo, formando módulos que el usuario gestionará avanzando secuencialmente en el cálculo de instalaciones renovables. Cada cuadro de diálogo, abarcará un tipo de sección específica del cálculo, las cuales irán desde la definición básica de la ubicación del entorno de construcción, pasando por parámetros técnicos de la instalación, hasta llegar al propio cálculo económico o de rentabilidad de la inversión.

El interior de cada cuadro, implementará estructuras de código a fin de facilitar el intercambio de información, con mínima necesidad de interpretación por parte del usuario, componiéndose tales elementos por cuadros de edición, botones de pulsación, listas desplegables y en general elementos que con independencia de la estructura programática que acarreen, generen un entorno intuitivo al que el usuario, sin considerar su nivel de conocimiento informático, reconozca de manera sencilla y consiga gestionarlos con facilidad.

INDICE

1.- Introducción

El presente proyecto aborda la problemática o necesidad de cálculo de sistemas productivos en instalaciones agropecuarias, a fin de poder constituir sistemas energéticos alternativos, que den servicio a este tipo de instalaciones, sobre las cuales cabe aplicarse tecnologías energéticas renovables que contribuyan a un ahorro económico a las empresas del sector, bien sea este propiciado por la propia producción energética, o por el ahorro en aras a una disminución de la dependencia de otro tipo de combustibles e incluso el aminoramiento de costes derivados del propio enclavamiento de las industrias agropecuarias (entornos aislados con bajo grado de electrificación o redes de distribución eléctrica).

En este sentido se ha desarrollado una herramienta informática diseñada en un lenguaje generalista como es el C++, en la cual se genera una interface gráfica de fácil utilización, pensada para el cálculo global de instalaciones de energías renovables, desde los conceptos básicos relacionados con el propio cálculo energético de las mismas, hasta análisis económico del proyecto de instalación.

Esta aplicación se considera que debe permitir al usuario, definir y contar con las cargas que se pueden presentar en un entorno agropecuario, estableciéndose que la alta tecnificación actual del sector, propicia que en entornos de esta índole, se encuentren cada vez puntos de consumo más diversos, quedando lejos la única presencia de motores y/o bombas, motivo por el cual el sistema se entiende debe permitir la fácil introducción de cualquier tipo de equipo, mediante libertad de definición y cálculo para el usuario.

Según esto, el programa indicado, se basa en un cálculo generalista de instalaciones de producción renovable, basado en pliegos técnicos de cálculo ampliamente utilizados en el sector, y permitiendo al usuario un fácil e intuitivo uso de la aplicación en el que se pueda incorporar cualquier tipo de equipamiento energético que se encuentre en entornos agropecuarios, siendo además que la modularidad del desarrollo efectuado, permite la fácil modificación, ampliación y orientación del aplicativo, no existiendo limites en cuanto a lo que se refiere a una especialización concreta de un entorno o sector definido.

Con respecto a la programación que se ha desarrollado, como lenguaje se ha elegido C++, considerándose idóneo para el desarrollo de aplicación graficas en entorno Windows, y resultando un lenguaje que soporta programación orientada a objetos (POO), con desarrollo de aplicaciones en menor tiempo al reusar código, permitiendo un lenguaje o código con tipos de datos fáciles de comprender. Así mismo se ha elegido como plataforma de desarrollo la aplicación Visual Studio, ofreciendo la misma al usuario una gestión del código, programación y modificaciones optimizada que facilita la evolución del diseño de la aplicación de manera estructurada y coherente.

Se ha buscado un entorno de muy fácil manejo, de manera que una vez el usuario tiene los datos básicos de la instalación a calcular, le permita introducirlos a través de un recorrido guiado a través de diversas pantalla, solventado cualquier necesidad de cálculo y requisitos de normativa, los cuales se embeben en el código,

haciendo que el usuario no tenga por qué tener un conocimiento profundo de las ecuaciones o requisitos necesarios en el cálculo de instalaciones de energías renovables.

2.- Objeto del proyecto

Tal como se ha indicado de manera previa, el objeto del presente proyecto, se centra en el cálculo de instalaciones de energías renovables, las cuales se orientan o conciben a complementar ciertas cargas energéticas, desde la definición de estas, hasta el cálculo de viabilidad y/o rentabilidad económica de la inversión. En este sentido, no se concibe una restricción de propósitos en el desarrollo de la presente aplicación, dado que el rango de necesidades en el sector agropecuario, es tan amplio y variado (estaciones de bombeo en regadío mediante invernadero, sistemas de alimentación de animales, ventilación y/o control de parámetros en granjas avícolas, porcinas, etc...), que se intenta la amplia funcionalidad del presente proyecto.

Así mismo, las características de la demanda energética, pueden variar radicalmente en cuanto a potencia necesaria, tipos de equipos, etc…, pero sin embargo el cálculo de las instalación de energía, permanecerá inalterable y será equivalente, esto es, el cálculo de producción para alimentar un ventilador, será idéntico si este está instalado en una granja avícola (asfixia de animales), o como ventilación para disminuir temperatura ambiente (refrescar salas o establos de granjas bobinas del sector lácteo), suponiendo esto, la necesidad de obtener un programa de cálculo global, pero que permita la definición abierta de equipos (el usuario debe poder introducir y definir de manera clara y simple sus equipos o puntos de consumo, con independencia de que clase sean estos y en qué tipo de explotación de emplacen).

2.1.- Características instalaciones agropecuarias

Las instalaciones agropecuarias actuales, presentan un número de variaciones indeterminado, en función de su orientación y características tecnológicas, así como el modelo constructivo y explotación, guardando poca relación explotaciones de baja extensión en invernadero para producción de lechuga, con las amplias instalaciones de procesado de productos para producción de vegetales en cuarta gama, las cuales requieren el propio procesado del producto en las instalaciones (presencia de secado en vacío, instalaciones de bombeo de alta potencia, etc…), ni que decir la presencia de instalaciones en sectores, avícolas, porcino y/o bobino.

Si bien es cierta la mencionada variación o necesidades, todas estas instalaciones presentan cierta problemática común:

- Se trata de instalaciones, ubicadas en entornos no urbanos, normalmente lejos de polígonos o en parajes agrestes, e incluso en zonas poco accesibles de montaña o parajes muy aislados, en los cuales el acceso a recursos energéticos puede ser complicado, o cuando menos, incrementar el coste de explotación, bien por el alto coste de la energía (generadores de gasóleo), o por la obra necesaria para obtener dicha energía (acometidas eléctricas aéreas sobre apoyos en recorridos de varios kilómetros).

- Los márgenes de la industria agropecuaria en el sector primario o de producción, se constriñen a precios de mercado a la baja, en un sector competitivo, de manera que, la subida de recursos energético básicos, como por ejemplo el gasóleo, puede afectar gravemente a la rentabilidad de la explotación, especialmente en Pymes o empresas familiares, siendo que de manera previsible, a lo largo de un periodo de tiempo amplio, los recursos energéticos fósiles, sufrirán elevados incrementos, tanto por lo finito de tales recursos, como la variación del mercado con fluctuación severas en función de cualquier tipo de parámetros político y/o macroeconómico.

- El recurso energético, debe ser fiable, dado que un paro en el suministro o alimentación de los sistemas puede conllevar graves pérdidas económicas por pérdida de producto, véase paradas de ventilación en granjas avícolas (decenas de miles de muertes de animales en breves periodos de tiempo sin ventilación), o imposibilidad de riego en fechas concretas en explotaciones agrícolas

2.2.- Sistemas consumidores de energía

De un simple análisis de la gran variedad de sectores agrarios, en que los que resulta imprescindible energía eléctrica para el funcionamiento y producción, cabe extrapolarse la abultada variedad de características que imponen cada uno de esto elementos, siendo así que explotaciones agrícolas, como norma general, presentarán una mayor incidencia del consumo de energía en estaciones de bombeo, mientras que explotaciones ganaderas sufrirán este incremento en alimentación de motores, destinados a movimiento de maquinaria.

Desde el punto de vista de consumo, no es factible describir un patrón regular de dichos consumos, variando este sustancialmente en función de la actividad agropecuaria desarrollada, así como el tamaño de la explotación, pudiendo fluctuar estos desde simples almacenes agrícolas cuyas únicas necesidades se refieren a iluminación, hasta complejas instalaciones de producción extensiva en invernadero, con elevadas potencias de bombeo, máquinas de procesado de mercancía (transportadores, equipos de lavado, etc…).

Figura 1: Ordeñadora robótica. Ref. Cat. DeLaval 2011

Si debe indicarse, que el actual incremento de coste de los recursos energéticos, así como las obvias necesidades de ahorro y cumplimiento de programas de eficiencia, de igual manera que la alta tecnificación del sector en aras a mejorar los resultados económicos, han hecho aparecer, junto con los sistemas tradicionales de producción (bombas, motores, etc…) un extenso campo de sistemas de control y regulación, los cuales parecían únicamente ligados al sector industrial tradicional, pero que se han

implantado de manera amplia en el sector agropecuario, haciendo variar el perfil de las instalaciones, mediante el uso de variadores, autómatas para control de producción, e incluso avanzados equipos robotizados adaptados a cualquier tipo de aplicación (apilamiento, ordeño ganado, etc…).

Con respecto al tipo de energía consumida en instalaciones agropecuarias, entendiéndose estas como las instalaciones o inmuebles en los que se enclava la actividad (descartadas maquinaria autopropulsada), la totalidad de la energía consumida es en forma de electricidad, siendo la obtención de la misma, de manera tradicional, en función del grado de aislamiento de la instalación, bien a través de red eléctrica, de manera general mediante acometidas eléctricas mediante tendido aéreo en media tensión y centros de trasformación propios, o enlaces a red en baja tensión (en caso de ubicarse cerca de núcleos urbanos o polígonos ganaderos), y a través de combustibles fósiles mediante generadores eléctricos (utilización de generadores diesel).

2.3.- Adaptabilidad fuentes renovables a sector

Visto el funcionamiento y consumos energéticos del sector, resulta evidente que la obtención de energía, como recurso imprescindible en producción del sector agropecuario, se puede adaptar de manera plena a fórmulas de producción mediante energías renovables, siendo las principales, la producción a través de sistemas fotovoltaicos y aerogeneradores, y obviamente pudiéndose aplicar otro tipo de fuentes que se vinculen con sectores más concretos (biogás en producciones ganaderas, energía térmica, etc…).

En el contexto agrícola actual, los ahorros de energía y las posibilidad de acceso a ésta, resultan problemáticas representativas y a tener en consideración en el desarrollo de una instalación de producción, con independencia de las características de la misma. Por un lado, el aumento del precio del petróleo provoca un aumento de los costes de producción de los productos agrícolas, mientras que los precios de venta de éstos no siguen la misma tendencia. Por otro lado, la búsqueda de ahorros y la transición hacia las energías renovables se integran en un enfoque que permite preservar el medio ambiente y luchar contra el cambio climático, así como controlar la fluctuación de mercado de otros productos energéticos.

Por tal circunstancia, hoy en día, el uso de energías renovables tales como fotovoltaica o la energía eólica, puede contribuir a la salvaguarda del entorno, ya que la agricultura está bien adaptada para este tipo de aplicaciones.

Los paneles fotovoltaicos permiten una producción de electricidad sin molestia: no ruido, no contaminación. Además, se puede instalar en todos los lugares que tienen una exposición al sol. Su esperanza de vida es superior a 20 años y no hay piezas en movimiento, y por tanto no hay desgaste y gastos de funcionamiento. Se pueden utilizar estos sistemas en almacenes aislados en montaña para alimentar el edificio de electricidad, o reducir costes o necesidad de aprovechamiento de otro tipo de producción energética (como energía de apoyo en la producción), poseyendo una gran adaptabilidad a cualquier ambiente agropecuario.

La instalación de un aerogenerador en una explotación agropecuaria puede ser muy ventajosa. Siendo despejados los terrenos agrícolas, o incluso dada la normal extensión de explotaciones de índole ganadera, existen emplazamientos disponibles para acoger un aerogenerador de baja potencia. Los aerogeneradores pueden producir, en tal tipo de aplicaciones, con viento medio, entre 10.000 y 65.000 KWh/año, siendo que este dato representa una producción superior a las necesidades energéticas de una explotación agrícola de tamaño medio. La instalación de un aerogenerador en una explotación agropecuaria es una inversión rentable y sostenible que permite a los agricultores sacar provecho de su propio yacimiento eólico y protegerse contra el aumento de precio de la energía.

No debe olvidarse, que los presente sistemas descritos, permiten aprovechar la energía en producción, o por otro lado conectarse a red eléctrica a fin de vender la producción, suponiendo esto un inversión mercantil de los sistemas productivos, o incluso efectuar instalaciones mixtas, que vendan el exceso de energía no utilizada por la instalación.

A modo de conclusión y con respecto a la integración de sistemas se producción energética basados en energía renovables, se puede decir, que tal tipo de tecnóloga se adapta a las características específicas del sector, incurriendo en la cobertura de las necesidades de las mismas y su demanda:

- La instalación de placas solares y/o aerogeneradores, dota a las diversas instalaciones de independencia, permitiéndoles cubrir su demanda, de manera total o parcial, pudiendo prescindir, y por tanto descartar los costes económicos, que suponen los gastos logísticos, de instalación y/o de elevado importe de recurso, derivados de la ubicación de instalaciones en entornos agrestes o zonas aisladas de instalación de estas explotaciones.

- La autogeneración, y el autoabastecimiento energético, o la mera posibilidad de la reducción de la dependencia de otro tipo de combustibles, como puede ser el gasóleo, hace interesante el uso de energías alternativas o renovables, repercutiendo esto de manera directa en los costes productivos de las instalaciones.

- Las instalaciones de generación propia, máxime si permiten acumulación como es el caso de la fotovoltaica, dotan a la instalación de un fondo de reserva energética, que permite entrar en funcionamiento los equipos, aun cuando haya imprevistos en los sistemas de producción clásicos (averías en líneas de distribución, generadores, etc…), pudiendo entrar en funcionamiento, en caso de emergencia o necesidad, supliendo cualquier tipo de hueco de energía, altamente lesivo en algunos tipos de instalaciones o épocas de producción.

A modo de conclusión, se establece como evidente, el amplio abanico de configuraciones, versatilidad y posibilidades que ofrece el uso de energías renovables en el sector agropecuario, así como la fácil integración de este tipo de sistemas, que por el propio ahorro energético que ofrecen suponen una inversión de rentabilidad manifiesta, en cuanto a lo económico se refiere, resultando también interesantes otros aspectos como posibilidad de independencia energética.

3.- Descripción genérica proyecto

El presente proyecto, se encamina a la integración de cálculo de instalaciones energéticas basadas en fuentes renovables (en la presente versión fotovoltaica), pensadas para una posible adaptación de éstas al sector agropecuario. Tal aplicativo, se compone de una interface basada en cuadros de dialogo que permitan un desarrollo simplificado, el cual mantiene latente en su programación la base teórica y estructura genérica del cálculo de este tipo de elementos.

El sistema se desarrolla en torno a los pliegos de condiciones técnicas de instalaciones, tanto conectada a red, como aislada, de IDAE, los cuales dan las nociones y metodología de cálculo, así como requisitos reglamentarios en las referentes potencias y tipos de instalaciones de tales características.

Por tal circunstancia, los pliegos técnicos especificados, se han integrado secuencialmente en cada una de las pantallas o módulos de proceso del cálculo de instalaciones fotovoltaicas, definiendo tales condicionantes los diferentes cálculos implícitos en la programación del proyecto.

3.1.- Características

La aplicación diseñada, se desarrolla en un sistema de entorno Windows para 32bits, permitiendo esto la funcionalidad de la misma en sistemas operativos tanto de 32 como de 64bits, siendo un aplicativo sencillo basado en cuadros de dialogo con elementos de entrada-salida de datos.

Desde la primera pantalla del aplicativo, se guiará al usuario en función de la instalación sobre la que desea hacer el cálculo, siendo que este en cualquier caso se restringe a la propia instalación de producción de energía, así como parámetros asociados a la misma, bien sus instalaciones (cálculo de cargas sobre cubiertas) o términos económicos y de rentabilidad asociados.

En ningún caso el presente programa entra a desarrollar sistemas de cálculo estructurales de instalaciones o soportes de las mismas, cálculo de retribución económica en función de los reales decretos existentes que regulan la venta de energía, ni otro tipo de análisis de las instalaciones, tales como diseños de ubicación, tendidos de cableados, centros de transformación, cuadros eléctricos o equivalentes.

En una primera pantalla de funcionamiento, se solicitará al usuario, decidir qué tipo de instalación desea calcular, siendo que a partir de esta, se iniciará un recorrido, en el que de manera general, se solicitarán características de la instalación (lugar de ubicación, latitud, etc…), procediéndose a solicitar los diferentes puntos de consumo de tal instalación, cálculo de potencia y características específicas, cálculo de solicitaciones de los elementos instalados (carga de viento, sobrecargas de cubiertas, etc…) y terminar efectuando un cálculo económico con respecto a parámetros de tomas de decisiones de inversiones tales como el VAN y el TIR. La totalidad de elementos introducidos y obviamente cálculos efectuados, podrán ser guardados y/o mostrados en informe en formato editable.

3.2.- Funcionalidad y orientación

Tal como se ha comentado de manera previa, el presente sistema se basa en un método de cálculo de instalaciones de energía renovables generalista, y que pueda dar servicio sea cual sea el tipo de instalación en el que se va a realizar la instalación, buscando así no encasillar el sistema en un único tipo de servicio o explotación.

Dada la generalidad del cálculo de sistemas de energías renovables, en el presente proyecto de tipo fotovoltaico, con respecto al mismo no hay diferencias entre un tipo de explotación u otra, buscándose la integración o personalización cada instalación, en la propia introducción de datos de éstas.

Mediante un entorno de descripción amplio, que permita al usuario introducir las cargas que desee o sus características, bien en forma de texto o referencias, se permitirá que cualquier elemento de una explotación agropecuaria sea definido. Por tal motivo se integrará esta introducción de datos en una única pantalla que permita ver la vista de los elementos asociados al consumo requerido:

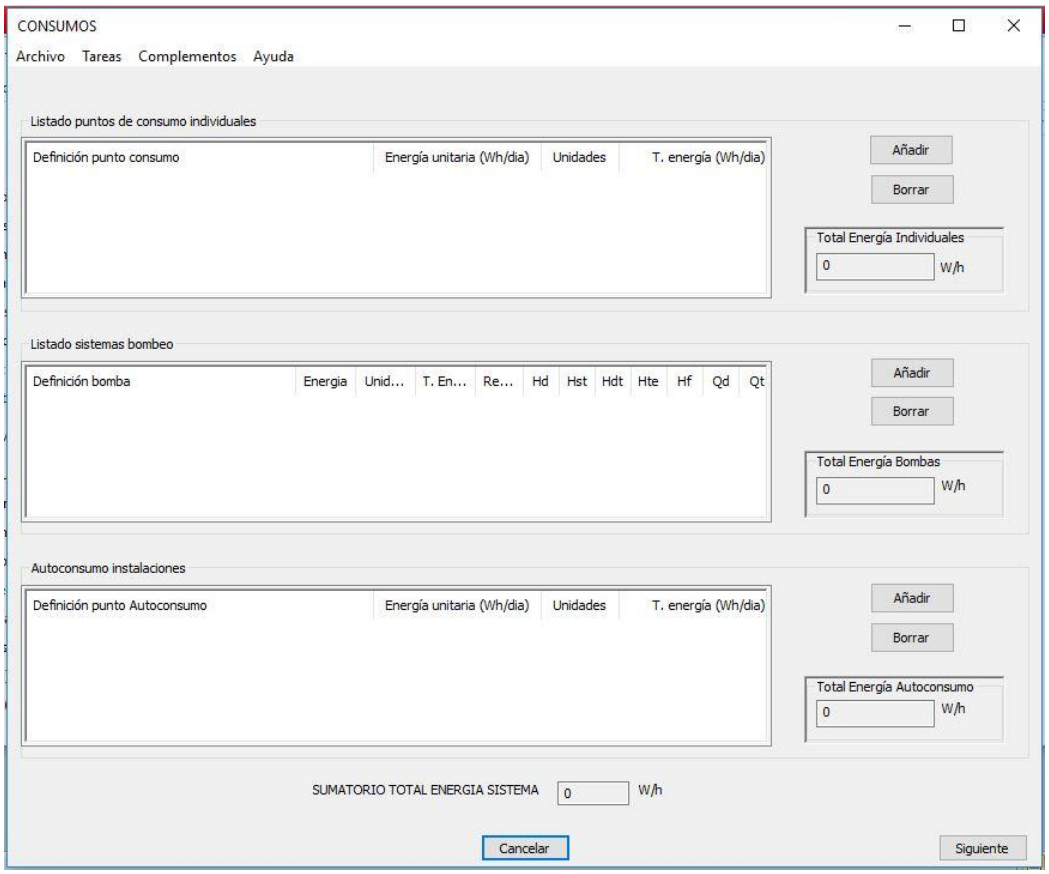


Figura 2: Cuadro dialogo consumos

Los puntos de consumo que el usuario puede introducir o seleccionar, son los referentes a:

- Cargas puntuales:

  Se busca la introducción generalista de cualquier elemento de la instalación, bien sea este un ventilador, un motor, un autómata o cualquier tipo de maquinaria específica, utilizándose para describir el mismo un cuadro de dialogo básico, en el que se permite una introducción textual de la definición del elemento, el número de elementos de una misma clase, y su potencia consumida.

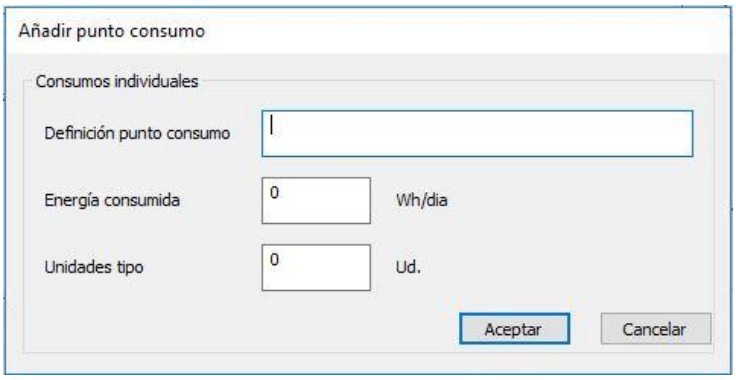


Figura 3: Cuadro dialogo introducir consumos

- Calculo de sistemas de bombeo:

  Dada la alta representatividad de las instalaciones de bombeo en el sector agropecuario, se inserta un capitulo propio. Tales instalaciones son fácilmente insertables en el capítulo anterior, pero se busca en el presente una descripción de estas, acorde a diversos parámetros del bombeo de un pozo acorde a las instrucciones de pliegos técnico de IDAE.

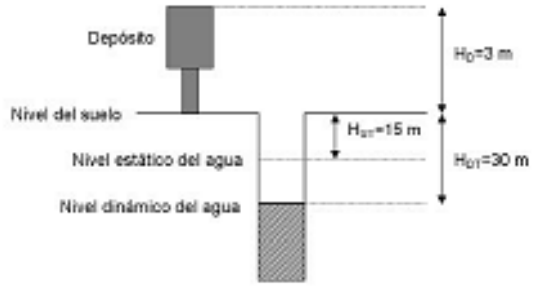


Figura 4: Esquema alturas pozo. Ref. IDAE

Estas especificaciones, indican la energía eléctrica consumida por una bomba, referidas a diferentes parámetros de la instalación, correspondiendo la ecuación general de cálculo a (cociente entre energía hidráulica del sistema y rendimiento del equipo de bombeo):

$$E_{MB}(Wh/dia) = \frac{E_H(Wh/dia)}{\eta_{MB}}$$

Obteniéndose la energía hidráulica de la siguiente expresión:

$$E_H(Wh/dia) = 2{,}725 \cdot Q_d(m^3/dia) \cdot H_{TE}(m)$$

$H_{TE}(m)$ es la altura teórica de bombeo, la cual es un parámetro ficticio que incluye características físicas del pozo y del depósito del cual realiza el bombeo, así como las perdidas por fricción de las tuberías del sistema y la variación de la altura del nivel del agua durante el bombeo, utilizándose para su cálculo la siguiente expresión:

$$E_{TE} = H_D + H_{ST} + \left(\frac{H_{DT} - H_{ST}}{Q_T} \right) \cdot Q_{AP} + H_f$$

En cualquier caso no se debe olvidar, que se ha previsto, que el usuario pueda no utilizar este tipo de introducción de datos, y que pueda referir la bomba, en una introducción de punto de consumo "puntual", definiendo únicamente la potencia del propio equipo, conocida a través de catálogo, siendo que el cuadro propio de introducción de bombas será:

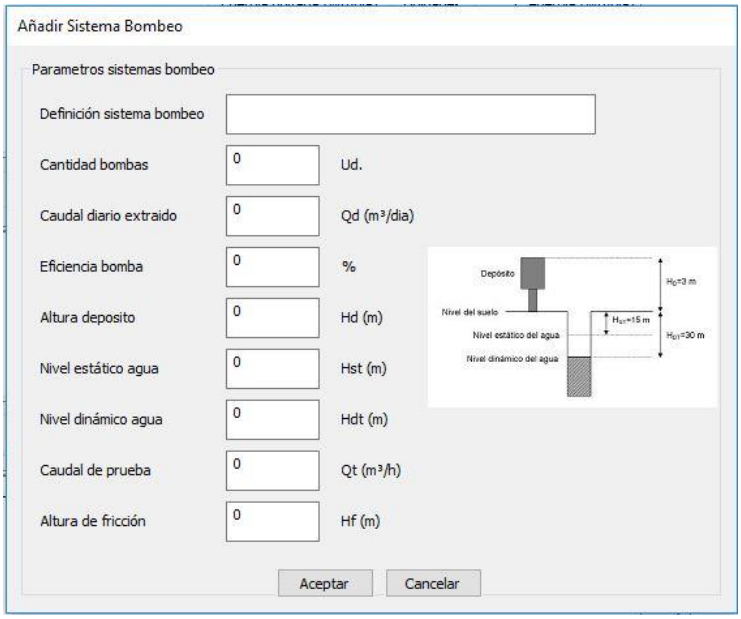


Figura 5: Dialogo insertar punto bombeo

- Puntos autoconsumo:

Finalmente y como tercer punto de consumo, se ha realizado un apartado específico para diferenciar el autoconsumo de los propios sistemas energéticos de producción de la instalación, tales como inversores, reguladores de carga y similares, siendo esto una contribución al consumo general de la instalación que debe estar considerado, y validándose su descripción según el cuadro de dialogo:

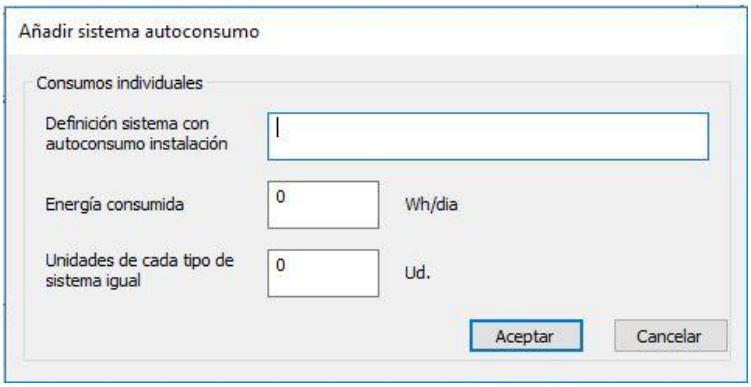


Figura 6: Cuadro dialogo punto autoconsumo

3.3.- Tipos de instalaciones

La presente versión del programa desarrollado, contempla dos tipos de instalaciones para cálculo de producción, siendo esta solares fotovoltaicas, y caracterizándose por su conexión o no a la red eléctrica.

A priori se considera, de manera obvia, que la energía producida en una instalación aislada de red, es exclusivamente destinada al autoconsumo, mientras que en instalaciones conectadas a red, se busca la venta completa de la energía producida, siendo esto muchas veces más rentable (venta de energía producida y compra de la requerida), que el aprovechamiento de la autoproducción. Si bien es cierto que a fin de realizar cálculos económicos, la presente aplicación, permitirá estimar o repartir la demanda y autoconsumo en el porcentaje que el usuario requerirá.

El aplicativo prevé la posibilidad de implementación del sistema a introducción del cálculo de energía eólica, o incluso aplicaciones mixtas fotovoltaica-eólica, no habiéndose implementado en la actualidad tales sistemas.

En lo referente al cálculo de estructura o sistemas de apoyo, únicamente se han contemplado la verificación de instalaciones de cubierta solar o suelo (por la dificultad de cálculo de estructuras, o peculiaridades específicas que presenta la instalación de seguidores).

3.4.- Base teórica desarrollo aplicación

La aplicación desarrollada se basa, en cuatro áreas definidas, siendo estas la introducción de cargas o necesidades de energía de las instalaciones (referidas en apartado anterior), el cálculo de la instalación energética de fuentes renovables, cálculo de cargas físicas sobre instalación y sobrecargas estructurales, y finalmente un análisis económico, siendo que la base teórica o de cálculo de cada uno de los siguientes apartados será:

a) Calculo de puntos de consumo:

Tal apartado ha sido previamente definido de manera amplia, siendo que el usuario tiene potestad para introducir diversos elementos o equipos consumidores de energía en la instalación en cálculo, basándose tal como se ha indicado, estos elementos en aspectos genéricos de potencia, desarrollándose el cálculo de bombas acorde a otras características propias de la instalación o pozo (balsa) de extracción.

b) Instalaciones solares fotovoltaicas:

El cálculo de tales instalaciones, se ceñirá a instalaciones conectadas a red y aisladas de red, contando cada una con sus peculiaridades, y siguiendo como base de desarrollo teórico los pliegos de condiciones técnicas de IDAE.

En lo referente a instalaciones aisladas de red, el sistema calculará o permitirá la introducción de los elementos que componen la misma, así como sus rendimientos, o características, siendo tales elementos los propios paneles solares fotovoltaicos, inversores, reguladores de carga y baterías o elementos de acumulación de energía.

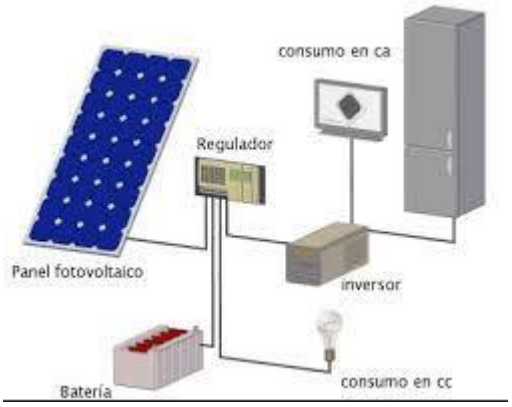


Figura 7: Equipos básicos instalación fotovoltaica aislada. Ref. Web Autoconsumamos

El cálculo de la energía diaria consumida, vendrá aportado por la introducción de cargas previas, con independencia de las características de las mismas, permitiendo esto el cálculo del consumo diario medio, necesario para requerimiento de almacenaje de las baterías.

$$L_D\left(Ah/dia\right)=\frac{E_D\left(Wh/dia\right)}{V_{NOM}\left(V\right)}$$

El dimensionado del sistema, requerirá del usuario la definición de dos parámetros básicos, tanto para la ejecución el sistema como para el posterior cálculo de pérdidas, siendo estos el ángulo de inclinación, que forma la superficie de los módulos con el plano horizontal y el ángulo azimut siendo este el que marca la proyección sobre el plano horizontal de la normal a la superficie del módulo y el meridiano del lugar.

Igualmente se deberá definir el periodo de diseño, comprendiendo este tres posibilidades, las cuales deberán ser seleccionadas en función del periodo más desfavorable para la instalación, como puede ser diciembre (siendo que la producción energética fotovoltaica es eminentemente inferior en tal mes), julio (resultando que en sistemas de riego este es el mes de mayor demanda energética), o en su caso todo el año (en instalaciones donde el consumo se vaya a considerar constante).

Con la combinación del ángulo de azimut, inclinación y el criterio de peor periodo (que corresponderá a la necesidad de diseño), se calcularán las perdidas por orientación, dado que existirá una perdida inherente a la diferencia entre opción de instalación optima, y en la que es posible realizar la instalación (en función de la casuística del lugar de ubicación de la misma), obteniéndose tales pérdidas de las expresiones:

$$FI = 1 - \left[1{,}2 \cdot 10^{-4} \cdot \left(\beta - \beta_{opt}\right)^2 + 3{,}5 \cdot 10^{-5} \cdot \alpha^2\right] \quad \text{para} \quad 15^o < \beta < 90^o$$

$$FI = 1 - \left[1{,}2 \cdot 10^{-4} \cdot \left(\beta - \beta_{opt}\right)^2\right] \quad \text{para} \quad 15^o < \beta < 90^o$$

para

Obteniéndose inclinaciones optimas de la diferencia o incremento para con la latitud del lugar que aporta:

Periodo de diseño	β_{opt}	$K = \frac{G_{dm}(\alpha = 0, \beta_{opt})}{G_{dm}(0)}$
Diciembre	$\phi + 10$	1,7
Julio	$\phi - 20$	1
Anual	$\phi - 10$	1,15

ϕ = Latitud del lugar en grados

Con respecto al cálculo de pérdidas por sombreo, este será igual, tanto en el caso de instalaciones conectadas a red, como las aisladas de red, expresándose tales perdidas como el porcentaje de radiación solar global que incidiría sobre la superficie de los paneles en caso de no existir sombra alguna.

El procedimiento de tal calculo consistirá en la comparación del perfil de obstáculos que afecta a la superficie de estudio, con el diagrama de trayectorias del sol, obteniéndose en primer lugar tal perfil de obstáculos, mediante la localización de estos elementos que afecten a la superficie de captación, en términos de sus coordenadas azimut y elevación. Tal

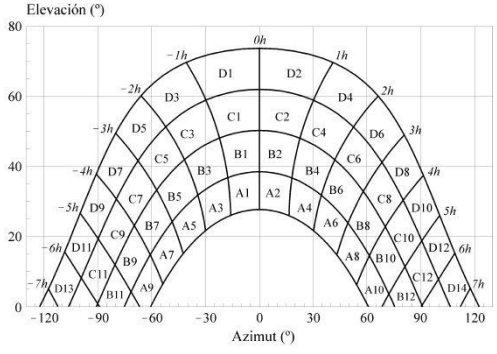


Figura 8: Diagrama pérdidas sombreo. Ref.IDAE

perfil de obstáculos se representara en el diagrama de perdidas correspondiente, en el que se muestra las bandas de trayectorias del sol a lo largo de todo el año y que se encuentran divididas en porciones delimitadas por horas solares. Cada una de esas zonas supondrá una contribución a la irradiación solar global anual que incide sobre los paneles solares y por tanto un obstáculo que cubre tal banda o porción de ésta, suponiendo una pérdida.

Según esto en el cálculo de pérdidas, se utilizarán tablas de valores específicos para determinas inclinaciones y ángulos azimut de la instalación, utilizándose siempre la más próxima a la realidad del proyecto en ejecución, siendo que de tales valores, se computarán los que producen sombreo, y el porcentaje en el que lo producen, obteniéndose así, las pérdidas por sombrero (FS).

El cálculo de la irradiación sobre el generador, y por extensión, de la energía que va a producir la instalación, se calculará teniendo en consideración las pérdidas de rendimientos por orientación y sombreo, según la expresión:

$$G_{dm}(\alpha,\beta) = G_{dm}(0)\cdot K\cdot FI\cdot FS$$

De donde el factor K, será obteniendo de manera empírica de tablas previas, o calculado en función de su expresión teórica como cociente entre irradiación de punto óptimo con irradiación global de la zona de instalación.

El dimensionado mínimo de la instalación solar, se realizara atendiendo a la expresión:

$$P_{mp,\min} = \frac{E_D\cdot G_{CEM}}{G_{dm}(\alpha,\beta)\cdot PR}$$

Se elegirá el tamaño del generador y del acumulador en función de las necesidades de autonomía del sistema, de la probabilidad de perdida de carga y cualquier otro factor a considerar, siendo la potencia como máximo un 20% superior al valor al dimensionado mínimo del generador, con autonómica mínima de 3 días y sin superar los acumuladores en 25 veces la corriente de cortocircuito en CEM del generador fotovoltaico, resultando la autonomía del sistema acorde a la expresión:

$$A = \frac{C_{20}\cdot PD_{\max}}{L_D}\cdot \eta_{inv}\cdot \eta_{rb}$$

Para el cálculo de instalaciones conectadas a red, se seguirá también el pliego técnico de IDAE, atendiendo a sus especificaciones características para el tipo de cálculo de instalaciones de energía, las cuales a priori se considera se destinarán íntegramente a venta, a fin de que el beneficio, mejore el ratio de eficiencia de la instalación agropecuaria.

El cálculo de la producción anual esperada, dependerá del valor medio de la irradiación mensual y anual diaria sobre un superficie horizontal, el valor medio de la irradiación sobre el plano del generador y el rendimiento energético de la instalación (PR), el cual, tal como sucedía en las instalaciones aisladas de red, depende de las pérdidas sobre el conjunto de la instalaciones (eficiencia cableados, dependencia de temperatura, etc…), siendo la estimación de la energía inyectada:

$$E_P = \frac{G_{dm}(\alpha, \beta) \cdot P_{mp} \cdot PR}{G_{CEM}}$$

Tal como se ha indicado, el cálculo de pérdidas por orientación y por sombras, seguirán el mismo principio de análisis que las instalaciones conectadas a red, siendo evidentemente elementos análogos y variando únicamente entre tal tipo de instalaciones el consumo primario que se da a la energía producida, bien sea este autoabastecimiento, o venta a red.

c) Cálculo de sobrecargas de cubierta:

El programa desarrollado hará una estimación de las cargas que producen en cubierta la instalación fotovoltaica desarrollada (únicamente se consideran instalaciones de cubierta o suelo, en ningún caso seguidores o sistemas con grado de desplazamiento). La base de cálculo de dichas instalaciones, se basará en los condicionantes de acción atmosférica de viento, así como peso propio, siguiéndose la base de desarrollo que se especifica o adapta a tal uso según Código Técnico de la Edificación (concretamente cálculos de marquesina a un agua).

Inicialmente se especificarán las acciones del viento sobre la estructura de los paneles, donde la acción del viento, genera una fuerza perpendicular a la superficie de cada punto expuesto, o presión estática que responde a la expresión:

$$q_e = q_b \cdot c_e \cdot c_p$$

Siendo la presión dinámica del viento:

$$q_b = 0{,}5 \cdot \delta \cdot v_b^2$$

Obteniéndose la velocidad del viento en función de la localización geográfica de la instalación, o en su caso del estudio que pueda caracterizar las medias de viento de una zona geográfica concreta:

Figura 9: Áreas viento. Ref. Código Técnico Edificación

El coeficiente dinámico del viento que dependerá del entorno y grado de aspereza del mismo, siendo la expresión matemática que lo define:

$$c_e = F \cdot (F + 7 \cdot k)$$

Con

$$F = k \cdot \ln(\max(z, Z)/L)$$

Siendo z la Altura del emplazamiento de instalación, y el resto de parámetros:

Tabla D.2 Coeficientes para tipo de entorno

	Grado de aspereza del entorno	Parámetro		
		k	L (m)	Z (m)
I	Borde del mar o de un lago, con una superficie de agua en la dirección del viento de al menos 5 km de longitud	0,15	0,003	1,0
II	Terreno rural llano sin obstáculos ni arbolado de importancia	0,17	0,01	1,0
III	Zona rural accidentada o llana con algunos obstáculos aislados, como árboles o construcciones pequeñas	0,19	0,05	2,0
IV	Zona urbana en general, industrial o forestal	0,22	0,3	5,0
V	Centro de negocios de grandes ciudades, con profusión de edificios en altura	0,24	1,0	10,0

Figura 10: Parámetros zonas aspereza terreno. Ref. Código Técnico Edificación

Finalmente el viento produce una sobrecarga sobre cada elemento superficial en la dirección de su normal, positiva (presión) o negativa (succión), dependiendo esta presión de la presión dinámica y el coeficiente eólico que dependerá de la

configuración de la construcción, la posición del elemento afectado por el viento y el ángulo de incidencia del viento en la superficie, pudiéndose obtener este valor del coeficiente eólico de tablas específicas.

Se estimaran también en un precálculo, las fuerzas soportadas por los perfiles estructurales de sujeción de los paneles solares, dependiendo las fuerzas de la solicitación de la acción del viento sobre los paneles y la longitud de los perfiles:

$$F = q_e \cdot l$$

La sobrecarga en la cubierta (valor del cual se da una orientación en la aplicación, no eximiendo esta del cálculo estructural que pudiera corresponder), quedará determinada por el peso propio de los paneles, y del peso de la estructura de aluminio de fijación.

d) Calculo de inversión:

Dada la manifiesta importancia que tiene el aspecto económico, tanto en el desarrollo de instalaciones, como en el análisis de la rentabilidad de la inversión, y por consiguiente en la toma de decisiones sobre la instalación y uso de los sistemas en estudio, se ha integrado la ejecución de un cálculo de rentabilidad, atendiendo a dos factores de análisis financiero, siendo estos el VAN y el TIR.

El valor actual neto, también conocido como valor actualizado neto es un procedimiento que permite calcular el valor presente de un determinado número de flujos de caja futuros, originados por una inversión. La metodología consiste en descontar al momento actual (es decir, actualizar mediante una tasa) todos los flujos de caja futuros o en determinar la equivalencia en el tiempo 0 de los flujos de efectivo futuros que genera un proyecto y comparar esta equivalencia con el desembolso inicial. Cuando dicha equivalencia es mayor que el desembolso inicial, entonces, es recomendable que el proyecto sea aceptado.

La fórmula que nos permite calcular el Valor Actual Neto es:

$$VAN = \sum_{t=1}^{n} \frac{V_t}{(1+k)^t} - I_0$$

La tasa interna de retorno o tasa interna de rentabilidad (TIR) de una inversión es la media geométrica de los rendimientos futuros esperados de dicha inversión, y que implica el supuesto de una oportunidad para "reinvertir". En términos simples, diversos autores la conceptualizan como la tasa de descuento con la que el valor actual neto (VAN) es igual a cero.

La TIR puede utilizarse como indicador de la rentabilidad de un proyecto: a mayor TIR, mayor rentabilidad. Se utiliza como uno de los criterios para decidir sobre la aceptación o rechazo de un proyecto de inversión. Para ello, la TIR se compara con una tasa mínima, el coste de oportunidad de la inversión. Si la tasa

de rendimiento del proyecto, expresada por la TIR, supera la tasa de corte, se acepta la inversión; en caso contrario, se rechaza.

$$VAN = -I_0 + \sum_{t=1}^{n} \frac{F_t}{(1+TIR)^t}$$

Se considera en las anteriores ecuaciones, la inversión en el periodo de tiempo de "n" años, siendo Vt (o Ft según el caso), los flujos de caja de cada uno de los años e I0 el importe de la inversión inicial o coste de ejecución del proyecto.

4.- Lenguaje de programación

El proyecto se ha desarrollado en programación sobre lenguaje C++, utilizando como plataforma de desarrollo Microsoft Visual Estudio V.15, incorporando bibliotecas MFC (Microsoft Foundation Classes), las cuales proporcionan soporte a una aplicación de entorno grafico que se ejecuta sobre sistema operativo Windows de 32bits.

4.1.- Generalidades C++

C++ es el lenguaje de programación multiobjetivo ideado en 1984 por Bjarne Stroustrup a partir del lenguaje C. Es un lenguaje imperativo orientado a objetos. En realidad un superconjunto de C, que nació para añadirle cualidades y características de las que carecía. El resultado es que sigue muy ligado al hardware subyacente, manteniendo una considerable potencia para programación a bajo nivel, pero se la han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción.

Una de las principales virtudes de este lenguaje es ser un lenguaje de propósito general que se adapta a múltiples situaciones, siendo que tal tipo de utilización es adecuada, en el caso del presente proyecto, a la ejecución de una aplicación Windows partiendo de cero, desarrollando el software a medida en función de los requisitos de la propia aplicación.

Las ventajas principales de la utilización de C++ se pueden considerar:

- Una de las grandes ventajas de C++ es la capacidad de tener abstracciones (tiene acceso a objetos y clases) muy eficientemente con respecto al hardware. Ya que C++ se construyó a partir de C, una de las características principales es que puede acceder directamente al hardware como lo hacen los sistemas operativos. Por eso la combinación es algo que es bueno para la el uso eficiente del hardware en general.

- Otro aspecto que es necesario para el buen funcionamiento de la infraestructura y el hardware es la estabilidad. Un lenguaje de programación debe ser estable.

- Se trata de un lenguaje de programación orientado a objetos.

- Lenguaje muy didáctico, que permite tener un conocimiento de la estructura general del mismo, así como disponer de múltiples tutoriales, libros y recursos didácticos para el desarrollo y comprensión del mismo.

- Es muy potente en lo que se refiere a creación de sistemas complejos, un lenguaje muy robusto.

4.2.- Programación orientada a objetos (POO)

La diferencia fundamental entre la programación procedural y la orientada a objetos está en la forma de tratar los datos y las acciones. En la primera aproximación ambos conceptos son cosas distintas, se definen unas estructuras de datos y luego se define una serie de rutinas que operan sobre ellas. Para cada estructura de datos se necesita un nuevo conjunto de rutinas. En la programación orientada a objetos los datos y las acciones están muy relacionadas. Cuando definimos los datos (objetos) también definimos sus acciones. En lugar de un conjunto de rutinas que operan sobre unos datos tenemos objetos que interactúan entre sí.

Un Objeto es una entidad que contiene información y un conjunto de acciones que operan sobre los datos. Para que un objeto realice una de sus acciones se le manda un mensaje. Por tanto, la primera ventaja de la programación orientada a objetos es la encapsulación de datos y operaciones, es decir, la posibilidad de definir tipos abstractos de datos.

De cualquier forma la encapsulación es una ventaja mínima de la programación orientada a objetos. Una característica mucho más importante es la posibilidad de que los objetos puedan heredar características de otros objetos. Este concepto se incorpora gracias a la idea de clase.

Cada objeto pertenece a una clase, que define la implementación de un tipo concreto de objetos. Una clase describe la información de un objeto y los mensajes a los que responde. La declaración de una clase es muy parecida a la definición de un registro, pero aquí los campos se llaman instancias de variables o datos miembro.

Cuando le mandamos un mensaje a un objeto, este invoca una rutina que implementa las acciones relacionadas con el mensaje. Estas rutinas se denominan métodos o funciones miembro. La definición de la clase también incluye las implementaciones de los métodos.

Se puede pensar en las clases como plantillas para crear objetos. Se puede decir que un objeto es miembro de una clase.

Podemos definir clases en base a otra clase ya existente. La nueva clase se dice que es una subclase o clase derivada, mientras que la que ya existía se denomina clase base. Una subclase hereda todos los métodos y atributos de su clase base, además de poder definir miembros adicionales (ya sean datos o funciones). Las subclases también pueden redefinir métodos definidos por su superclase.

A la hora de programar con objetos crearemos una jerarquía de clases, definiendo una clase raíz que da a todos los objetos un comportamiento común (en el presente proyecto prioritariamente se utiliza *CDialogEx* de cara a tener una interface basada en cuadros de dialogo). La clase raíz será una clase abstracta, ya que no crearemos instancias de la misma. En general se debe definir una clase (o subclase de la clase raíz) para cada concepto tratado en la aplicación.

Para soportar el concepto de programación orientada a objetos se incluyen diversos mecanismos:

- Declaración de clases (usando la palabra class).

- Paso de mensajes a los objetos (llamadas a funciones).

- Protección de los métodos y atributos de las clases (definición de métodos privados, protegidos y públicos).

- Soporte a la herencia y polimorfismo (incluso se permite la herencia múltiple, es decir, la definición de una clase que herede características de dos clases distintas).

4.3.- Clases y objetos

La idea de clase junto con la sobrecarga de operadores, permite al usuario diseñar tipos de datos que se comporten de manera similar a los tipos estándar del lenguaje. Esto significa que debemos poder declararlos y usarlos en diversas operaciones como si fueran tipos elementales, siempre y cuando esto sea necesario. La idea central es que los tipos del usuario sólo se diferencian de los elementales en la forma de crearlos, no en la de usarlos.

Las clases no sólo nos permiten crear tipos de datos, sino que nos dan la posibilidad de definir tipos de datos abstractos y definir sobre estos nuevas operaciones sin relación con los operadores estándar del lenguaje. Introducimos un nuevo nivel de abstracción y comenzamos a ver los tipos como representaciones de ideas o conceptos que no necesariamente tienen que tener una contrapartida a nivel matemático, sino que pueden ser conceptos relacionados con el mundo de nuestro programa. Así, podremos definir un tipo "*inversión*" y un tipo "*diseño generador*" en un programa cálculo de fuentes renovables como el presente. Sobre estos tipos definiremos una serie de operaciones que definirán la interacción de las variables (*objetos*) de estos tipos con el resto de entidades de nuestro programa.

Otra idea fundamental a la hora de trabajar con clases es la distinción clara entre la definición o interface de nuestros tipos de datos y su implementación, esto es, la distinción entre qué hace y cómo lo hace. Una buena definición debe permitir el cambio de la estructura interna de nuestras clases sin que esto afecte al resto de nuestro programa. Deberemos definir los tipos de manera que el acceso a la información y operaciones que manejan sea sencillo, pero el acceso a las estructuras y algoritmos

utilizados en la implementación sea restringido, de manera que una modificación en estos últimos sólo sea percibido en la parte de nuestro programa que implementa la clase.

Una clase es un tipo de datos que se define mediante una serie de miembros que representan atributos y operaciones sobre los objetos de ese tipo. Hasta ahora conocemos la forma de definir un tipo de datos por los atributos que posee, lo hacemos mediante el uso de las estructuras.

C++ nos da la posibilidad de incluir funciones definidas como miembros de un tipo específico de clase. A estas funciones se les denomina miembros o métodos, y tienen la peculiaridad de que sólo se pueden utilizar junto con variables del tipo definido.

Con respecto a los objetos que se emplean en una clase, podemos clasificar éstos en cuatro tipos diferentes según la forma en que se crean:

- Objetos automáticos: son los que se crean al encontrar la declaración del objeto y se destruyen al salir del ámbito en que se declaran.
- Objetos estáticos: se crean al empezar la ejecución del programa y se destruyen al terminar la ejecución.
- Objetos dinámicos: son los que se crean empleando el operador new y se destruyen con el operador delete.
- Objetos miembro: se crean como miembros de otra clase o como un elemento de un array.

4.4.- Entorno de desarrollo

El sistema de desarrollo Microsoft Visual C++, utilizado en la programación del presente proyecto, es un entorno integrado de desarrollo para aplicaciones de C y C++ sobre diversas plataformas.

Para el desarrollo de programas, Windows prevé rutinas que permiten utilizar componentes con menús, cuadros de dialogo y barras de desplazamiento entre otros. Así mismo, se puede manipular el ratón, el teclado y diversos periféricos para entrada y salida de datos. Visual C++ es un entorno de programación que combina la programación orientada a objetos de C++, con el sistema de desarrollo de aplicaciones graficas en Windows.

Visual C++ incluye además de las herramientas específicas para generar el código de programación, una colección completa de clases (MFC), que permiten crear de forma intuitiva aplicaciones en entorno Windows manejar sus componentes.

Microsoft Foundation Classes o MFC es un conjunto de clases interconectadas por múltiples relaciones de herencia, que proveen un acceso más sencillo a las API de Windows. Con el paso del tiempo Microsoft Foundation Classes se ha convertido en la implementación estándar de la industria para la creación de aplicaciones gráficas en plataformas Windows.

MFC proporciona a C++ para Windows macros de tratamiento de mensajes (a través de mapas de mensajes), las excepciones en tiempo de ejecución e identificación del tipo RTTI, la serialización y la creación de instancias de clases dinámicas. Las macros para manejo de mensajes dirigidos a reducir el consumo de memoria, evitando el uso gratuito de tablas virtuales y también para proporcionar una estructura más concreta para Visual C++, suministrado herramientas para editar y manipular el código sin necesidad de analizar el lenguaje completo.

El entorno de desarrollo empleado de visual C++, presenta como características relevantes:

- Proporciona una biblioteca MFC que da soporte a objetos Windows (ventanas, cajas de dialogo, etc…).
- Es un entorno de desarrollo integrado (editor de texto, compilador, depurador, explorador de código fuente…).
- Aporta asistentes para desarrollo de aplicaciones (generación de cuadros de diálogo, menús, etc…).

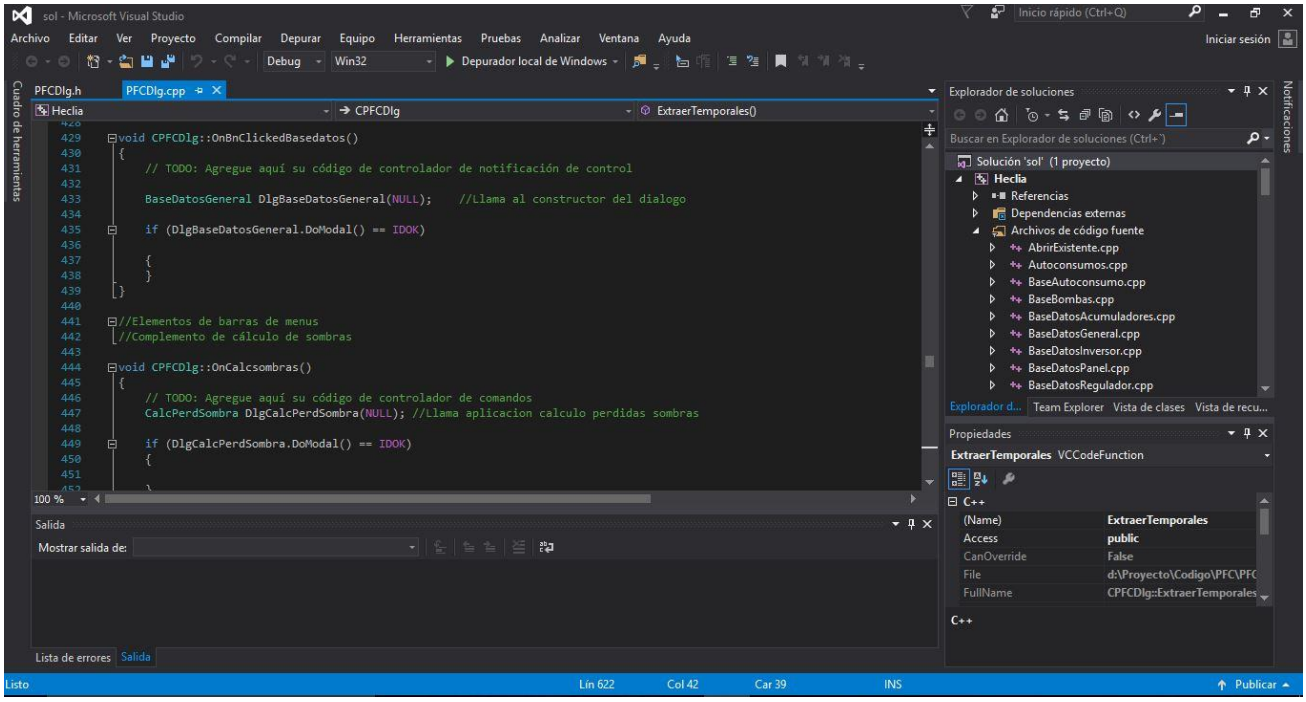


Figura 11: Interface Visual Studio C++ V15

4.5.- Tipos de archivo (.h, .cpp)

El soporte a la programación modular en C++ se consigue mediante el empleo de algunas palabras clave y de las directivas de compilación. Lo más habitual es definir cada módulo mediante una cabecera (un archivo con la terminación .h) y un cuerpo del módulo (un archivo con la terminación .cpp). En el archivo cabecera ponemos las

declaraciones de funciones, tipos y variables que queremos que sean accesibles desde el exterior y en el cuerpo o código definimos las funciones públicas o visibles desde el exterior, además de declarar y definir variables, tipos o funciones internas a nuestro módulo.

Si queremos utilizar en un módulo las declaraciones de una cabecera incluimos el archivo cabecera mediante la directiva "*#include*". También en el fichero que empleamos para definir las funciones de una cabecera se debe incluir el .h que define.

Existe además la posibilidad de definir una variable o función externa a un módulo mediante el empleo de la palabra *extern* delante de su declaración (función básica empleada en el presente proyecto a fin de comunicar los diferentes módulos o clases empleadas entre sí para intercambio de datos y variables globales.

En el archivo de código (*.cpp*), se definirán las acciones a realizar, mediante llamadas a funciones e insertando el código que realiza la ejecución de las diferentes acciones y/o cálculos. En este archivo se implementarán los elementos que hagan las llamadas para la gestión de la información, permitiéndose la incorporación de múltiples funciones, no solo referentes a las acciones a ejecutar en cada clase, sino globales para entrada/salida de datos, acciones de seriación, etc…

Los "*recursos*" son un archivo de texto que permite al compilador manejar objetos tales como imágenes, sonidos, cursores de ratón, cuadros de diálogo, etc. Microsoft Visual Studio permite crear un archivo de recursos particularmente fácil, proporcionando las herramientas necesarias en el mismo entorno utilizado para programar. Esto significa que no se tiene que utilizar una aplicación externa para crear o configurar un archivo de recursos. A continuación se presentan algunas características importantes relacionadas con los recursos.

- Los recursos son elementos de la interfaz que proporcionan información al usuario.
- Los mapas de bits, iconos, barras de herramientas y los cursores son todos los recursos.
- Estos elementos pueden ser manipulados para realizar una acción como la selección de un menú o introducir datos en el cuadro de diálogo.
- La aplicación utiliza varios recursos que se comportan de forma independiente el uno del otro, estos recursos se agrupan en un archivo de texto con la extensión "*.rc*".

Con respeto a los archivos de sistema y declaración de clases (archivos .h) de la aplicación desarrollada, estos se generaran siguiendo el esquema:

Declaración de variables globales mediante el uso de la definición "*extern",* mediante la cual se define una variable, del tipo que sea, como utilizable por cualquier clase de la aplicación, de tal manera que tanto desde la clase en la que se ha definido, como desde las restantes, se tendrá acceso y libertad de manipulación sobre tal variable. Esto es, una vez asignado un valor, este permanecerá inalterable y disponible en

cualquier parte del programa, hasta que sea modificado en alguna sección específica del código.

```
#pragma once
#include "afxcmn.h"

extern CString DescripcionBaseAux;
extern CString ModeloPanelBaseAux;
extern double PotPanelBaseAux;
extern double ICCPanelBaseAux;
extern double IMPPanelBaseAux;
extern double VMPPanelBaseAux;
extern double VCAPanelBaseAux;
extern double BasePanelBaseAux;
extern double AltoPanelBaseAux;
extern double PesoPanelBaseAux;
extern double CostePanelBaseAux;
extern bool DatoPanel;
```

Se declara a continuación la clase que se genera, relacionándola con la clase base de la que se deriva (en el ejemplo CDialoEx), incorporándosele los elementos implícitos de la misma tales como la macro que le corresponda a la clase (DECLARE_DYNAMIC agrega la capacidad de tener acceso a la información en tiempo de ejecución sobre una clase de objeto al derivar una clase de *CObject*), el constructor y destructor de la clase, y las funciones que le puedan ser implícitas.

```
// Cuadro de diálogo de BaseDatosPanel

class BaseDatosPanel : public CDialogEx
{
	DECLARE_DYNAMIC(BaseDatosPanel)

public:
	BaseDatosPanel(CWnd* pParent = NULL);   // Constructor estándar
	virtual ~BaseDatosPanel();		// Destructor estándar

	virtual void OnFinalRelease();
	virtual void Serialize(CArchive& ar);
	virtual BOOL OnInitDialog();

// Datos del cuadro de diálogo
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_BASEDATOSPANELES };
#endif

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // Compatibilidad
con DDX/DDV

	DECLARE_MESSAGE_MAP()
	DECLARE_DISPATCH_MAP()
	DECLARE_INTERFACE_MAP()
```

Finalmente se declararan en cada clase sus funciones controladoras de mensajes (*afx_msg*), que serán los encargados de gestionar las acciones de funcionamiento desencadenadas del accionamiento de elementos del cuadro de dialogo de la clase (movimientos de ratón, captación de pulsación de botones, centralización de foco sobre controles, perdidas de foco, etc….), así como la definición de las variables miembro de la clase (variables propias internas de cada clase que no son visualizables, compartidas ni manipulables por otras clases).

```
public:

	// Variable de control y funcionamiento base datos paneles
	afx_msg void OnBnClickedAnadirpaneles();
	afx_msg void OnBnClickedBorrarpaneles();
	afx_msg void OnBnClickedGuardarpanel();
	afx_msg void OnNMClickDatospaneles(NMHDR *pNMHDR, LRESULT *pResult);
	afx_msg void OnBnClickedButton1();

	CListCtrl m_DatosSalidaPaneles;
	int Elemento, ElementoPanel;
	CString DescripcionBase, ModeloPanelBase, PotPanelBase, ICCPanelBase,
	IMPPanelBase, VMPPanelBase, VCAPanelBase, BasePanelBase, AltoPanelBase,
	PesoPanelBase, CostePanelBase;
};
```

Igualmente en los archivos principales o cuerpo de código y funcionamiento de cada clase (.cpp), el desarrollo o estructura general de cada uno de ellos, conllevará la siguiente implementación (sin analizar el código interno de cada función que será analizado en el *Anexo 2*):

En primer lugar, se declararan o inicializarán las relaciones con otras clases, declarándose los archivos de cabecera de cada una de ellas, sobre las que el archivo implementado, o su clase, ha de reconocer, operar e incorporar.

```
#include "stdafx.h"
#include "PFC.h"
#include "BaseDatosPanel.h"
#include "afxdialogex.h"
#include "IntroDatosPaneles.h"
#include "BaseDatosGeneral.h"
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ostream>
#include <stdio.h>
#include <conio.h>
#include <iostream>
#include <cstring>

using namespace std;
```

Posteriormente se lanzará el dialogo y las funciones del mismo, inicializando las variables globales que le vayan a afectar (variables de todo el programa y no miembro especifico de la clase que implementa), inicializa funciones de construcción, destructor,

etc… y mensajes o controladores de acciones y componentes o controles montados en el cuadro de dialogo concreto de la clase.

```
// Cuadro de diálogo de BaseDatosPanel
IMPLEMENT_DYNAMIC(BaseDatosPanel, CDialogEx)

//inicialización variables externas
//datos introducción paneles base datos
CString DescripcionBaseAux, ModeloPanelBaseAux;
double PotPanelBaseAux,ICCPanelBaseAux, IMPPanelBaseAux, VMPPanelBaseAux;
double VCAPanelBaseAux, BasePanelBaseAux, AltoPanelBaseAux, PesoPanelBaseAux;
double CostePanelBaseAux;
bool DatoPanel;

BaseDatosPanel::BaseDatosPanel(CWnd* pParent /*=NULL*/)
	: CDialogEx(IDD_BASEDATOSPANELES, pParent)
{
	EnableAutomation();
}

BaseDatosPanel::~BaseDatosPanel()
{
}

void BaseDatosPanel::OnFinalRelease()
{
	CDialogEx::OnFinalRelease();
}

void BaseDatosPanel::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_DATOSPANELES, m_DatosSalidaPaneles);
}

BEGIN_MESSAGE_MAP(BaseDatosPanel, CDialogEx)
ON_BN_CLICKED(IDC_ANADIRPANELES, &BaseDatosPanel::OnBnClickedAnadirpaneles)
ON_BN_CLICKED(IDC_BORRARPANELES, &BaseDatosPanel::OnBnClickedBorrarpaneles)
ON_BN_CLICKED(IDC_GUARDARPANEL, &BaseDatosPanel::OnBnClickedGuardarpanel)
ON_NOTIFY(NM_CLICK, IDC_DATOSPANELES, &BaseDatosPanel::OnNMClickDatospaneles)
ON_BN_CLICKED(IDC_BUTTON1, &BaseDatosPanel::OnBnClickedButton1)
END_MESSAGE_MAP()

BEGIN_DISPATCH_MAP(BaseDatosPanel, CDialogEx)
END_DISPATCH_MAP()

// Nota: suministramos compatibilidad con IID_IBaseDatosPanel para admitir
enlaces de seguridad de tipos
//  desde VBA.  Este IID debe coincidir con el GUID asociado a la interfaz
Dispinterface
//  del archivo .IDL.

// {8B6CB760-10DC-4B02-9A5F-583FA8B04FAE}
static const IID IID_IBaseDatosPanel =
{ 0x8B6CB760, 0x10DC, 0x4B02, { 0x9A, 0x5F, 0x58, 0x3F, 0xA8, 0xB0, 0x4F,
0xAE } };

BEGIN_INTERFACE_MAP(BaseDatosPanel, CDialogEx)
INTERFACE_PART(BaseDatosPanel, IID_IBaseDatosPanel, Dispatch)
END_INTERFACE_MAP()
```

Finalmente en el cuerpo del archivo, se realizarán llamadas a las funciones asociadas a cada control o herramienta, siendo que en el cuerpo de tales funciones serán implementadas para que en caso de actuación en el aplicativo, se ejecuten las instrucciones que contienen.

```
void BaseDatosPanel::OnBnClickedAnadirpaneles()
{
   // TODO: Agregue aquí su código de controlador de notificación de control
}

void BaseDatosPanel::OnBnClickedBorrarpaneles()
{
}

void BaseDatosPanel::OnBnClickedGuardarpanel()
{
}

void BaseDatosPanel::OnNMClickDatospaneles(NMHDR *pNMHDR, LRESULT *pResult)
{
       LPNMITEMACTIVATE pNMItemActivate
       reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR);

       *pResult = 0;
}

void BaseDatosPanel::Serialize(CArchive& ar)
{
}

void BaseDatosPanel::OnBnClickedButton1()
{
}
```

4.6.- Estructura general aplicación

Atendiendo a los criterios anteriormente especificados en lo que refiere al uso de clases y objetos en la presente aplicación, cabe definirse la estructura particular y de generación de código que se ha empleado en el desarrollo del proyecto.

La generación de código se ha desarrollado mediante una estructura de clases dependientes en árbol, siendo la base principal en la que se basan las estructuras de dialogo de la interface del usuario, precisamente en este tipo de elemento, "*cuadros de dialogo*", gestionándose por tanto estos desde la clase raíz adecuada al uso, siendo esta "*CDialogEx*, de la cual se derivaran las diversas clases generadas.

La clase CDialog, propicia la funcionalidad necesaria para la generación de cuadros de dialogo, sobre los cuales posteriormente se insertarán los automatismos y controles necesarios para el desarrollo del software generado. CDialogEx, se deriva directamente de CDialog, siendo funciones equivalentes, pero pudiéndose gestionar en la primera de ellas funciones relacionadas con las vistas, tales como color de fondo de cuadro, imágenes de fondo del mismo, etc…, así como mejora la funcionalidad del uso

de diversos tipos de herramientas sobre el dialogo, circunstancia imprescindible en el presente proyecto.

En el proyecto se ha generado una clase por cada uno de los diálogos o pantallas que el usuario va a utilizar a las que puede llamar, derivándose a su vez todas ellas de la clase base comentada.

```
// Cuadro de diálogo de CPFCDlg
class CPFCDlg : public CDialogEx
{
DECLARE_DYNAMIC(CPFCDlg);
friend class CPFCDlgAutoProxy;

// Construcción
public:
CPFCDlg(CWnd* pParent = NULL);    // Constructor estándar
virtual ~CPFCDlg();

// Datos del cuadro de diálogo
enum { IDD = IDD_PFC_DIALOG };
```

Según esto cada cuadro de dialogo al que el usuario convoque, o por el que navegue dentro de la aplicación generada, corresponderá a una clase concreta que lo generara, controlará y albergará todos su elementos (funciones, variables miembro, etc…). Dado que en la aplicación, cada cuadro de dialogo, llama a otros elementos, esto es, funciones que hacen llamadas a cuadros de dialogo y por extensión a sus clases, se harán llamadas mediante dialogo modales.

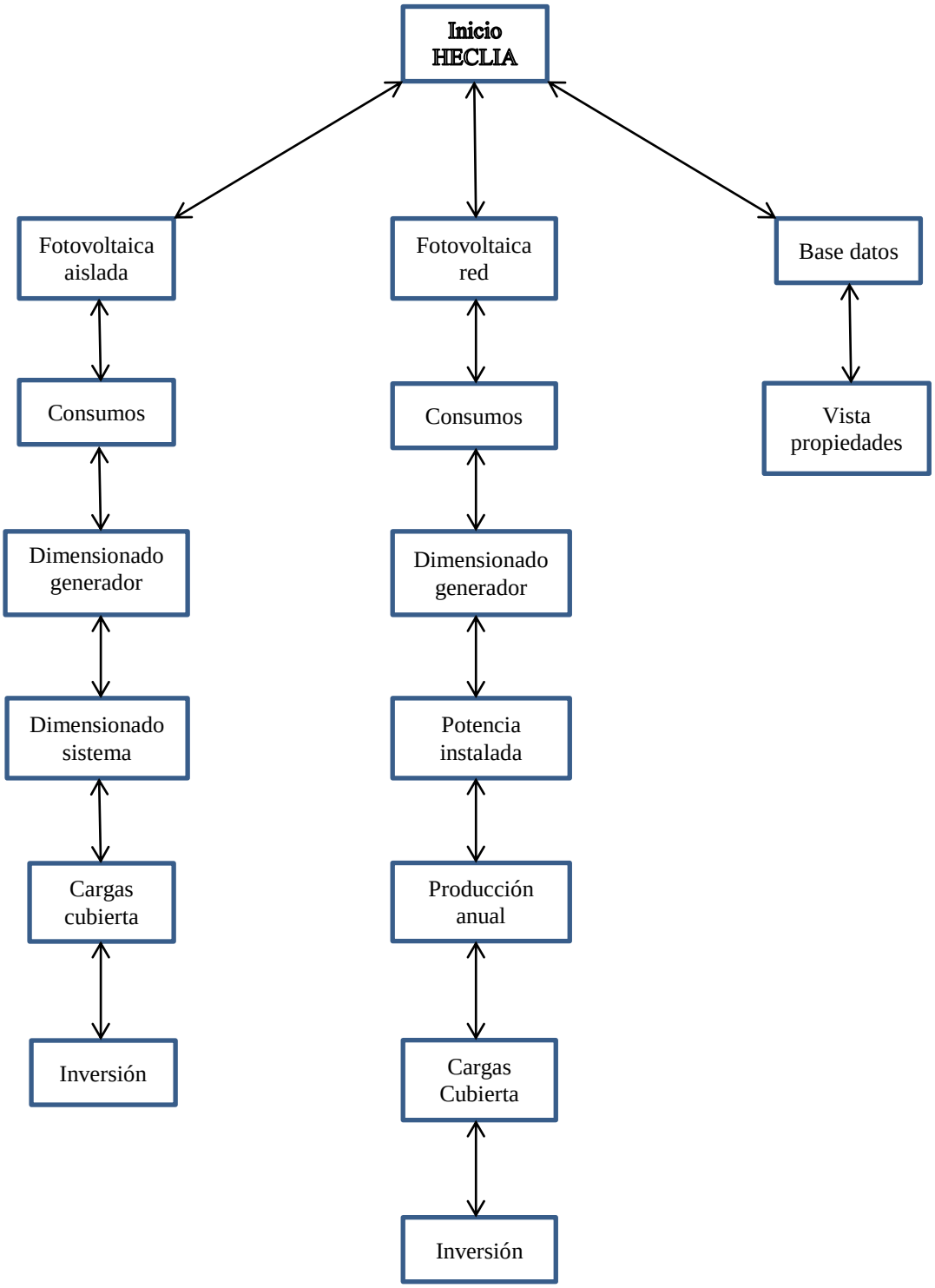
La clase base utilizada, permite dos tipos de llamadas a cuadros de dialogo, modales, y no modales, siendo que los primeros exigen al usuario, que conteste, acepte o gestiones el cuadro para continuar con la aplicación, y los segundos permanecen en pantalla disponibles para su uso, pero permiten seguir manipulando el resto del proceso. En el presente caso, se ha elegido el uso de cuadros modales, a fin de que el usuario, gestione cada pantalla de modo obligado, antes de pasar a la siguiente.

La invocación o llamadas entre clases de cada una de las visualizaciones de la aplicación responderán a la estructura:

```
void CPFCDlg::OnCalcsombras()
{
	// TODO: Agregue aquí su código de controlador de comandos
	CalcPerdSombra DlgCalcPerdSombra(NULL); //Llama aplicación calculo
	perdidas sombras

	if (DlgCalcPerdSombra.DoModal() == IDOK)
	{
	}
}
```

Las relaciones entre clases que se han desarrollado en el aplicativo, responderá a la siguiente estructura, a través de la que se genera el árbol que las compone:



En la presente aplicación, además de haber usado como clase base del desarrollo, o de la cual reciben herencia el resto de clases, se utilizan en módulos o elementos asociados otras clases como *CDocument* (encargada de gestionar archivos o ficheros en

tareas de búsqueda/escritura de archivos) y *CMapStringToOb* (encargada de generar mapas de objetos para almacenar elementos asociados entre sí).

5.- Elementos entorno aplicación

Tal y como se ha comentado a lo largo de la presente memoria, la aplicación desarrollada tiene una base fundamentada en cuadros de dialogo, los cuales se invocan a través de llamadas modales, que bloquean el cuadro hasta que el usuario continua con la aplicación, cancela la acción o retrocede en el menú.

Estos cuadros de dialogo, compondrán cada uno de ellos una clase diferente, y contendrán todos los controles, o elementos necesarias para que sirvan de interface de intercambio de información, datos introducidos por el usuario, y datos aportados por el sistema en función de las operaciones matemáticas internas programadas.

De igual manera el programa desarrollado permitirá toda funcionalidad, y características, comunes y ordinarias que se consideran habituales en entornos de trabajo software Windows, tales como vistas, guardado de información, navegación por formularios, etc…

5.1.- Base desarrollo aplicación

Como base de la presente aplicación, se muestra las bases o plantillas de cuadro de dialogo. En la generación del proyecto se ha diseñado una plantilla diferente para cada vista, con sus características que se ajustan a los objetivos a obtener de cada una de las secciones del programa, pensando en incrustar sobre la misma una serie de controles, que respondan a acciones, cuya programación implícita se asocie a las necesidades de cálculo del sistema energético en estudio, presentando como base la teoría expuesta en aparatados previos del proyecto.

Según esto, para la generación de cada vista, se da comienzo insertando un recurso, u objeto en blanco, para lo cual se elegirá la generación de una plantilla en blanco para cuadro de dialogo, sin referencias o funcionalidad alguna incorporada, más allá del propio código embebido que aportan las bibliotecas MFC para gestión de la vista.

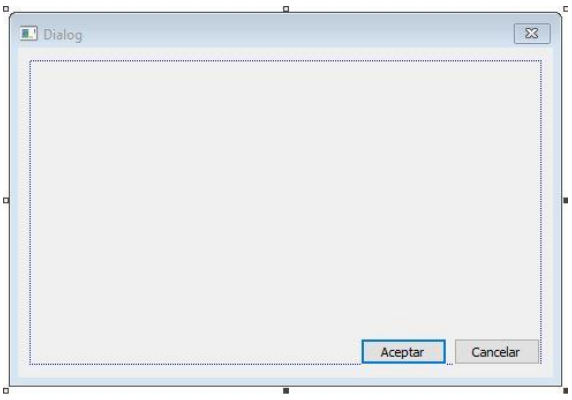


Figura 12: Vista base cuadro dialogo

Sobre esta plantilla se incorporaran secuencialmente, en función de las necesidades de desarrollo, los controles que se consideren necesarios, según las herramientas disponibles y preprogramadas (en cuando a forma, que no código de funcionamiento) que ofrece el, propio entorno de desarrollo de Visual C++.

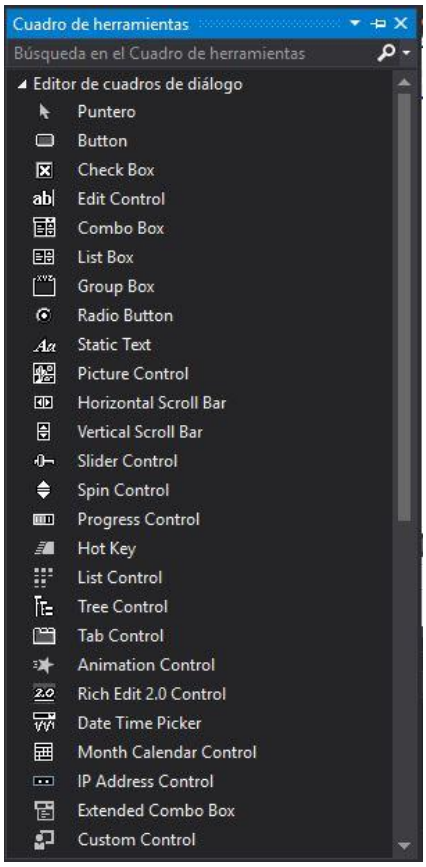


Figura 13: herramientas/controles

Mediante este desarrollos, se obtendrá una ventana o cuadro de dialogo personalizado, mediante el cual se pretende que el usuario sea capaz de gestionar las necesidades de cálculo previstas en la aplicación, presentando la vista en uso, todas aquellas características que se han definido de manera previa en el proceso de diseño.

Cada una de las herramientas insertadas, compone un control, el cual será referenciado por un código de identificación, que lo personalizará, a fin de que sea reconocido por el sistema y forme una entidad particular, sobre la que se puedan definir unos eventos (puesta de foco sobre el control, perdida de foco, pulsación sobre elemento, doble pulsación de ratón, etc…) y en su caso una variables miembro vinculada a tal control, que propicie el intercambio de información entre el propio control y el código desarrollado por el usuario.

Así mismo cada control permitirá una personalización del mismo, en cuanto a forma y características, permitiendo esto variaciones en su funcionamiento interno o visualización. De tal manera e insertando secuencialmente las herramientas con controles necesarios, obtendremos la vista de programa (como se ha comentado una vista por cada pantalla de uso diseñada, y asociada a cada clase del programa).

Generando una clase por cada pantalla grafica desarrollada, obtendremos las diferentes plantillas del programa base del presente proyecto, pudiéndose hacer llamadas entre vistas (según diálogos modales como se ha definido previamente), e incorporando a cada una de las clases, las funciones miembro que responden a cada evento de la plantilla ejecutada, en función de que el usuario interactúe para con cada herramienta insertada en la vista.

5.2.- Tipos controles utilizados

Para hacer que una aplicación responda a eventos (acciones sobre controles insertados en vista) uniremos a cada uno de los controles, el código con el que deben responder a cada evento de interés que se produzca sobre ellos. Este código asociado a cada control y escrito para un determinado evento o suceso, recibe el nombre de procedimiento conducido por un evento. En la presente aplicación, estos procedimientos

serán funciones miembro de la clase sobre la que se esté definiendo el cuadro de dialogo concreto.

Los diferentes procedimientos, serán invocados en función del tipo de control utilizado, el cual proveerá al conjunto del programa de unas características o posibilidades inherentes a cada control.

Los controles utilizados en la aplicación, e insertados encada cuadro de dialogo serán:

a) Texto estático y cuadros de grupo.

   Ambos elementos se emplean de manera organizativa, más que con una funcionalidad enlazada a código o acción alguna del programa, siendo que los primeros servirán para insertar etiquetas o nombres de controles, mientras que los segundos proveerán de un cuadro con texto estático editable para agrupar en su interior varios controles o elementos de una misma sección de la aplicación.

   Ambos controles no están habilitados para interactuar en modo alguno, sino que son meramente textos indicativos, bien simples que indican o marcan el nombre de otros controles, a modo de información estática al usuario o como en el mencionado caso de los cuadros de grupo, con un recuadro para englobar varios elementos análogos o con similar funcionalidad.

b) Controles de botón.

   Dentro de los controles de botón, se considerarán de tres tipos, resultando estos, botones propiamente dichos, botones de radio y cuadros de marcar, siendo la aplicación de cada uno de los tipos particular y específicamente aplicada a las necesidades de la interface.

   Un botón simple representa una orden que se ejecuta siempre que se pulse éste e invoca la acción que debe llevar a cabo. En la presente aplicación, se unen dichos controles a un evento o gestor de suceso, siendo el más utilizado el de pulsación, incorporándose en el código el mensaje *BN_CLICKED*, que hace referencia a que se ha pulsado el control y deriva a la ejecución del código a la acción que se ha definido para ese evento o pulsación en concreto.

   Con respecto a los botones de radio, en un grupo de los mismos solo puede estar marcado un elemento a la vez. Dichos botones son mutuamente excluyentes, esto es, la marcación de uno implica el desmarcado del resto, para que así el usuario pueda elegir una opción de entre un grupo de estas.

   Dado que el objetivo de este tipo de botón, es seleccionar una opción de entre un grupo, en algún momento del código, va a ser necesario que el código recupere la opción seleccionada, esto es, el usuario pulsa uno de los elementos en la interface, y antes de poder gestionar el evento que le corresponda o código implementado, será necesario recuperar o verificar cual ha sido la opción pulsada. Para conseguir esto,

se asocia una variable el grupo de botones de radio, de manera que en función del valor de ésta, sepamos que botón esta seleccionado.

Sirva a modo de ejemplo la implementación de este tipo de botón en la zona del aplicativo que selecciona, el porcentaje de sombra en cada sector para el cálculo de pérdidas a sombreo, donde las opciones de pulsación del usuario serán:

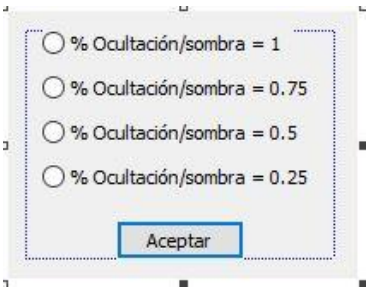


Figura 14: Cuadro dialogo selección % ocultación

Así como el código asociado al control especificado (en el cual en función de la selección realizada se asignará un valor a las variables que integrará o se sumará al cálculo de las pérdidas):

Tal y como se ha comentado en el propio constructor de la clase se inicializarán variables miembro para cada elemento u opción inicializados a falso:

```
Ocultacion::Ocultacion(CWnd* pParent /*=NULL*/)
	: CDialog(IDD_OCULTACION, pParent)
	, Sombra1(false)
	, Sombra2(false)
	, Sombra3(false)
	, Sombra4(false)
```

Y automáticamente se asociará cada botón u opción a un evento de marcado, excluyente unos para con otros, de modo que si se marca uno se desmarquen los demás

```
BEGIN_MESSAGE_MAP(Ocultacion, CDialog)
	ON_BN_CLICKED(IDC_RADIO1, &Ocultacion::OnBnClickedRadio1)
	ON_BN_CLICKED(IDC_RADIO2, &Ocultacion::OnBnClickedRadio2)
	ON_BN_CLICKED(IDC_RADIO3, &Ocultacion::OnBnClickedRadio3)
	ON_BN_CLICKED(IDC_RADIO4, &Ocultacion::OnBnClickedRadio4)
	ON_BN_CLICKED(IDOK, &Ocultacion::OnBnClickedOk)
END_MESSAGE_MAP()
```

Y cuando se presente cada evento, se asigne a la variable de porcentaje de sombra, un valor determinado, el cual al cerrarse el cuadro (mediante botón de pulsación estándar), cargará o empleará el valor de la variable (*VSombra*), que se hayan validado en función de la selección realizada.

```
void Ocultacion::OnBnClickedRadio1()
{
	VSombra = 1;
}
void Ocultacion::OnBnClickedRadio2()
{
	VSombra = 0.75;
}
void Ocultacion::OnBnClickedRadio3()
{
	VSombra = 0.5;
}
void Ocultacion::OnBnClickedRadio4()
{
	VSombra = 0.25;
}
void Ocultacion::OnBnClickedOk()
{
	VSombra;
	VSombraAux = VSombra;
	CDialog::OnOK();
}
```

En referencia a los cuadros para marcar, este tipo de control, permite al usuario seleccionar una opción, varias o ninguna, siendo su estilo normal un rotulo estático junto a cuadro de marcado. Tal como sucedía en los botones de radio, su objetivo es marcar opciones (aunque no excluyentes entre ellas), de manera que mediante el mismo sistema que los botones de radio, deberá ser recuperado si el botón está marcado o no, a fin de actuar en consecuencia en el interior del control, siendo el código que implementa tal acción:

A modo de ejemplo uno de los controles de la aplicación implementados con este sistema (botón de marcado de cálculo de inversión en el que se selecciona si la instalación es asilada de red o no), se asignará una variable miembro, la cual cuando se ejecute la revisión del control, permitirá, en función de su valor (true o false), elegir la acción asociada, recuperándose su valor mediante un puntero al propio botón de pulsación (*CButton* m_TipoInstalacion = (CButton*)GetDlgItem(IDC_TIPOINSTAL)).

```
void Inversión::OnEnKillfocusPorcentajecobertura()
{
   UpdateData(TRUE);
   CButton* m_TipoInstalacion = (CButton*)GetDlgItem(IDC_TIPOINSTAL);

   if (!m_VerificacionInstalacion)
      {
       m_CoberturaDemanda = 0.00;
       Q = m_Necesidades*m_VentaEnerg;
      }

   else if (m_VerificacionInstalacion)
      {
       m_CoberturaDemanda;
       Q = (m_Necesidades*m_CompraEnerg) -(m_Necesidades*m_CompraEnerg*(1-
       (m_CoberturaDemanda/100)));
}
```

c) Control de cuadro de edición.

Los controles de cuadro de edición se utilizan para admitir texto y mostrar texto. Los caracteres, los números y los símbolos habituales existentes en teclado se pueden insertar y visualizar en un cuadro editable.

En la presente aplicación, el cuadro de edición es utilizado para intercambiar la información y datos requeridos, bien en sentido de introducción por parte del usuario, o de salida de información procedente de la ejecución de funciones internas del programa.

Así mismo en la aplicación, se ha desarrollado sistemas que permitan la realización de modificación, borrado o cambio de información reflejada en dichos controles, o en su caso bloquear la manipulación o edición por parte del usuario para que los controles sean solo de lectura, o incluso modificaciones en ciclo de ejecución, para que el control pueda pasar de editable a solo lectura en función de las selecciones que realice el usuario.

d) Controles de lista.

Los controles de lista pueden ser de cuatro tipos: cuadros combinados, controles de lista, cuadros de lista y arboles (utilizándose en el presente proyecto los dos primeros). Cada uno de estos elementos, se adecua a un requisito concreto de la programación asociada a la interface del usuario desarrollada.

Con respecto a los cuadros combinados, de las posibilidades que ofrecen al control de la interface, se han ejecutado a modo de lista desplegable para realizar una opción por parte del usuario de entre múltiples posibles, quedando tras la selección visible únicamente el texto o contenido de tal opción, como si de un cuadro de edición se tratase.

La funcionalidad utilizada según esto será un modo "*Drop List*", que combina un texto estático con un botón y un cuadro de lista. El cuadro de lista solo es visible cuando se actúa con el control y el usuario no podrá modificarlo al ser sus opciones estáticas (texto no editable) y deber elegir una concreta.

Este control, al igual que los demás especificados en la presente memoria, se asociará a una variable miembro específica, de la clase del elemento que se está manipulando, siendo que al elegir el usuario una de las opciones disponibles en la lista, tal variable adquirirá este valor concreto, pudiéndose ejecutar selecciones mediante comparación del texto o elemento elegido, para con las opciones disponibles en la toma de decisiones, ejecutando según esto un código u otro.

```
void DGenerador::OnCbnSelchangePeriodo()
{
	if (m_SelecPeriodo == _T("Diciembre"))
	{
		m_AlfaOpt = 0;
		m_BetaOpt = m_Latitud + 10;
		m_K = 1.7;
	}

	if (m_SelecPeriodo == _T("Julio"))
	{
		m_AlfaOpt = 0;
		m_BetaOpt = m_Latitud - 20;
		m_K = 1;
	}

	if (m_SelecPeriodo == _T("Anual"))
	{
		m_AlfaOpt = 0;
		m_BetaOpt = m_Latitud - 10;
		m_K = 1.15;
	}

	UpdateData(FALSE);
}
```

Con respecto al control de lista, el método para poblar y manejar el mismo es diferente al anterior. Mediante una edición de este tipo de control, tendremos una pantalla de texto, la cual dividiremos en columnas para reflejar información en cada una de ellas perteneciente a un mismo objeto, siendo posible incluso la colocación de un marcador o indicador en el encabezado de cada una de esas columnas.

La clase MFC utilizada para encapsular este tipo de control es *CListCtrl*, siendo que esta clase presenta las funciones asociadas a la adición y eliminación de elementos de este tipo de control.

En la clase del cuadro de dialogo en la que se inserta este tipo de control se empieza incluyendo la definición del control en la función *OnInitDialog()*, la cual es invocada justo tras la creación del cuadro o interface, y se encarga de cargar los conceptos o estructuras iniciales del control.

Se definirán el número de columnas numerándolas desde el 0 como valor inicial hasta el último elemento que queramos definir, siendo que tal definición efectuada mediante la función *InsertColumn()*, define 4 parámetros, siendo estos el número de columna, el texto que refleja en el encabezado, la alineación del texto, y la anchura de la columna.

```
BOOL Consumos::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	//se inician las columnas del control de lista de datos puntuales
	m_DatosPuntuales.InsertColumn(0, _T("Definición punto consumo"),
	LVCFMT_LEFT, 290);
	m_DatosPuntuales.InsertColumn(1, _T("Energía unitaria (Wh/dia)"),
	LVCFMT_CENTER, 140);
	m_DatosPuntuales.InsertColumn(2, _T("Unidades"), LVCFMT_CENTER, 65);
	m_DatosPuntuales.InsertColumn(3, _T("T. energía (Wh/dia)"),
	LVCFMT_CENTER, 150);

	return TRUE;  // return TRUE unless you set the focus to a control
}
```

Cuando el usuario inserte un dato compuesto en la tabla (elementos conformado por varios elementos asociados a un mismo objetos, cuyos valores se distribuirán en función de la correspondencia por las diferentes columnas del control), éste se situará en el control en función del orden establecido y definición (orden descendente, ascendente, por inserción, etc…). Los datos introducidos, tendrán una etiqueta que los definirá y por cuyo nombre serán reconocidos, cargándose posteriormente el resto de valores, como asociados a tal etiqueta.

Según esto en función del dato a introducir, se empleará un cuadro de dialogo especifico, para introducir los valores representativos, los cuales de manera obvia serán introducidos en la variable que les corresponda, tipo entero, texto, etc…

El proceso de generación de los datos que se deben mostrar en el cuadro, una vez generadas las columnas en la inicialización, consistirá en asignar un valor numérico al nombre o elementos representativos del grupo (el que marcará la dirección de la línea y a través de la cual se relacionarán el resto de datos introducidos como referencias al mismo), utilizándose posteriormente la función *SetItemText()*, para dar salida por pantalla en el control a cada elemento, definiendo esta función tres factores: el elemento principal del objeto definido, la posición de ocupa en la fila y el valor de salida que se debe mostrar.

Cabe comentarse que en la lista únicamente se puede representar texto, de tal manera que si los datos que queremos introducir son valores numéricos, previamente deberemos convertirlos a texto antes de poder manipularlos. Obviamente a la inversa será un procedimiento idéntico, dado que un texto recuperado de la lista, deberá ser convertido al valor numérico que deseemos para poder operar con él.

```
//añadir fila al control de lista
              int nItem;
              nItem = m_DatosPuntuales.InsertItem(0, definicionPuntualAux);

//dar formato a los archivos numéricos para salir en pantalla como texto
              CString t;
              CString y;
              CString u;
              t.Format(_T("%f"), energiaPuntualAux);
              y.Format(_T("%d"), unidadesPuntualAux);
              u.Format(_T("%f"), totalPuntualAux);

// insertar en cada columna de una misma línea cada uno de los datos de los
elementos
              m_DatosPuntuales.SetItemText(nItem, 0, definicionPuntualAux);
              m_DatosPuntuales.SetItemText(nItem, 1, t);
              m_DatosPuntuales.SetItemText(nItem, 2, y);
              m_DatosPuntuales.SetItemText(nItem, 3, u);
```

La recuperación de los datos de la lista, se efectuará mediante un evento en el cual la pulsación sobre un elemento concreto de la lista, lanza un procedimiento para capturar los datos intrínsecos a tal elementos, donde se producirá la captura de la referencia del elemento seleccionado (*GetHotItem*), y tal como se ha indicado se reconvertirán los valores de texto, a la clase de valor requerida para su manipulación fuera del control.

```
void Consumos::OnNMClickVistapuntual(NMHDR *pNMHDR, LRESULT *pResult)
{
//Función para capturar la pulsación del ratón en la lista de autoconsumos
LPNMITEMACTIVATE pNMItemActivate = reinterpret_cast<LPNMITEMACTIVATE>(pNMHDR);

       NM_LISTVIEW* pNMListView = (NM_LISTVIEW*)pNMHDR;

       //obtención del número de elemento o referencia lista
       Elemento = m_DatosPuntuales.GetHotItem();

       //Recuperación de los valores de los elementos de lista en formato texto
       Nombre = m_DatosPuntuales.GetItemText(Elemento, 0);
       EUnitaria = m_DatosPuntuales.GetItemText(Elemento, 1);
       Unid = m_DatosPuntuales.GetItemText(Elemento, 2);
       EnTotal = m_DatosPuntuales.GetItemText(Elemento, 3);

       //conversión de los valores de texto a valores numéricos para su uso
       EnergUPuntual = _wtof(EUnitaria);
       Ud = _wtoi(Unid);
       EnergTPuntual = _wtof(EnTotal);

       *pResult = 0;
}
```

Una vez recuperados todos los datos del elemento seleccionado en la vista del control, la manipulación de tales elementos será trivial, tanto para hacer operaciones, guardar los datos en disco duro, o borrar el elemento, función sencilla mediante el uso de:

```
// borrar el elemento de la lista
m_DatosPuntuales.DeleteItem(Elemento);
```

e) Hojas de propiedades.

Una hoja de propiedades no es un control como tal, sino que se trata de un cuadro de dialogo para mostrar las propiedades o diferentes características de elementos, utilizándose en nuestro caso para la generación de la base de datos de materiales, en el cual se utiliza una página base, sobre la que se incrustan cuatro tipos de controles de lista, independientes entre ellos, pero integrados en una única vista.

Al igual que los controles de lista, este tipo de elementos requerirá la inicialización del diseño en el propio momento de generar o invocar al cuadro de dialogo, efectuándose de manera análoga al caso previo, pero en ese caso declarando que controles o elementos se encapsularán en cada una de las páginas de la hoja de propiedades, así como cuál será el que se visualice inicialmente por defecto, esto es, no se definirán las filas y columnas de los controles, sino en sí que controles se utilizarán en cada vista.

```
BOOL BaseDatosGeneral::OnInitDialog()
{
	CDialogEx::OnInitDialog();

	m_CtrlBaseDatosGeneral.InsertItem(0, _T("Paneles fotovoltaicos"));
	m_CtrlBaseDatosGeneral.InsertItem(1, _T("Inversores"));
	m_CtrlBaseDatosGeneral.InsertItem(2, _T("Acumuladores"));
	m_CtrlBaseDatosGeneral.InsertItem(3, _T("Regulador Carga"));

	CRect rect;
	m_CtrlBaseDatosGeneral.GetClientRect(&rect);
	m_DatosPanel.Create(IDD_BASEDATOSPANELES, &m_CtrlBaseDatosGeneral);
	m_WndShow = &m_DatosPanel;
	m_DatosInversor.Create(IDD_BASEDATOSINVERSORES,
	&m_CtrlBaseDatosGeneral);
	m_WndShow = &m_DatosInversor;
	m_DatosAcumulador.Create(IDD_BASEDATOSACUMULADORES,
	&m_CtrlBaseDatosGeneral);
	m_WndShow = &m_DatosAcumulador;
	m_DatosRegulador.Create(IDD_BASEDATOSREGULADOR,
	&m_CtrlBaseDatosGeneral);
	m_WndShow = &m_DatosRegulador;

	//comienza activada por defecto la ventana 0
	iSelControl = 0;
	m_DatosPanel.ShowWindow(SW_SHOW);
	m_WndShow = &m_DatosPanel;


	return TRUE;  // return TRUE unless you set the focus to a control

}
```

La selección de cada una de las páginas de la hoja de propiedades, conllevará la activación de la vista que se incruste en tal hoja y la ocultación del resto. El control y actuación sobre los elementos de cada una de las páginas de la hoja, se gestionará y controlarán, en función de los controles específicos que alberge, empleándose para esto las técnicas previamente comentadas.

```
void BaseDatosGeneral::OnTcnSelchangeBasedatos(NMHDR *pNMHDR, LRESULT
*pResult)
{
	iSelControl = m_CtrlBaseDatosGeneral.GetCurSel();

	if (iSelControl == 0)
	{
		m_DatosPanel.ShowWindow(SW_SHOW);
		m_DatosInversor.ShowWindow(SW_HIDE);
		m_DatosAcumulador.ShowWindow(SW_HIDE);
		m_DatosRegulador.ShowWindow(SW_HIDE);
		m_WndShow = &m_DatosPanel;
	}
	else if (iSelControl == 1)
	{
		m_DatosInversor.ShowWindow(SW_SHOW);
		m_DatosPanel.ShowWindow(SW_HIDE);
		m_DatosAcumulador.ShowWindow(SW_HIDE);
		m_DatosRegulador.ShowWindow(SW_HIDE);
		m_WndShow = &m_DatosInversor;
	}
	else if (iSelControl == 2)
	{
		m_DatosInversor.ShowWindow(SW_HIDE);
		m_DatosPanel.ShowWindow(SW_HIDE);
		m_DatosAcumulador.ShowWindow(SW_SHOW);
		m_DatosRegulador.ShowWindow(SW_HIDE);
		m_WndShow = &m_DatosAcumulador;
	}
	else if (iSelControl == 3)
	{
		m_DatosInversor.ShowWindow(SW_HIDE);
		m_DatosPanel.ShowWindow(SW_HIDE);
		m_DatosAcumulador.ShowWindow(SW_HIDE);
		m_DatosRegulador.ShowWindow(SW_SHOW);
		m_WndShow = &m_DatosRegulador;
	}

	*pResult = 0;
}
```

f) Picture control

El presente control se utilizará para encastrar en una vista (en el propio interior del control), imágenes de mapas de bits, siendo que este control da soporte y marco para la instalación de fotografías y elementos gráficos estáticos sobre la vista de la interface.

5.3.- Entrada/salida datos

Dado que el presente aplicativo utiliza de manera masiva cuadros de edición de texto (descritos en apartados previos), es necesario e imprescindible el intercambio de información para con éstos, bien a modo de escritura y captación de los datos introducidos, bien a modo de salida de información en éstos elementos para ser mostrados al usuario.

Tal acción será llevada a cabo cuando el usuario ejecute alguna acción que ordene a la aplicación el intercambio de información, tal como se ha dicho en entrada o salida de la misma, respondiendo tal intercambio a un evento.

A fin de que esta tarea pueda ser realizada, cada cuadro de edición, con su identificación particularizada, será asociado a una variable miembro perteneciente la clase en la que se haya definido el control concreto, resultando que la información se cargará en tal variable, o se extraerá de la misma para ser reflejada en el cuadro de diálogo. Tal viable de manera obvia será de la clase o tipo de la información que se espere intercambiar con el cuadro de edición (*int, bool, doublé, CString…*).

Para intercambiar datos con estas variables y por tanto reflejarlos se utilizará la función *UpdateData()*, siendo que permitirá dos opciones de intercambio, siendo estas *UpdateData(TRUE)* para leer datos del control hacia la variable asociada y *UpdateData(FALSE)*, para extraer datos de la variable hacia el control.

```
void DGenerador::OnEnKillfocusFs()
{
	m_FSombreo;
	UpdateData(TRUE);

	// salida perdidas sombreo
	m_PerdidasSom = 1 - m_FSombreo;

	//salida de Gdm de diseño según ángulos elegidos
	m_GdmDis = m_Gdm0*m_K*m_FIrradiacion*m_FSombreo;
	GAux = m_GdmDis;
	UpdateData(FALSE);
}
```

El método de intercambio de datos anteriormente indicado, se utiliza en la vista de la interface, siendo que sin embargo de manera estandarizada, es necesaria la extracción de información que sea transportable a otro formato, utilizándose para esto archivos de texto a modo de informe, los cuales sean editables y se puedan imprimir o guardar.

La generación de tales archivos de salida consistirán en:

- Creación del archivo de salida generando un nombre y un tipo de archivo, así como fijando sus características. Mediante la declaración de la clase *ios*, definiremos el tipo de archivo (*out* para escritura, *in* para lectura, *binari* para binario, *ate* para comenzar la escritura al final del archivo, etc…).

```
//crea el archivo TXT en el que se va a generar el archivo
ofstream GeneracionArchivosSalida;
GeneracionArchivosSalida.open("InformeSalida.txt",ios::out);
```

- Escritura en el archivo generado, siendo que esta lectura puede ser de varias clases en función de los elementos que se quieran interponer, correspondiendo estos a:

  - Texto estático que se reflejara tal como lo predefinamos en el código programado, escribiéndose todos los elementos que se entrecomillen como texto estático, aceptando en la escritura elementos indicadores de retorno de carro, paso de líneas, etc… ("\n").

    La escritura en el archivo se realiza mediante el operador de indicación "<<", siendo que en el caso de que quisiéramos leer el contenido de un archivo el operador de extracción seria el inverso ">>".

```
//salidas de texto de modulo diseño sistema aislado de red

GeneracionArchivosSalida << "DISEÑO SISTEMA DE GENERACION FOTOVOLTAICA AISLADA
DE RED\n";
GeneracionArchivosSalida << "\n";
```

  - En el caso de que se desee escribir texto, procedente de una variable asociada (en c++ tipo CString o Char), no se podrá pasar directamente al archivo. Esto es, el texto deberá ser convertido previamente a un formato transferible a archivos de salida

    Esta acción se realizará obteniendo inicialmente la longitud de la cadena de texto a procesar en función del texto de la misma (al programar en formato UNICODE, será necesaria gestionar esta acción mediante dos comandos de captura de longitud *GetLength()* de una variable *CString* y de esta obtener la longitud convertida en un *TCHAR*).

    Posteriormente con la acción *write*, pasando como parámetros el texto de salida en *CString* y su longitud, se podrá procesar o escribir el texto en el archivo TXT. Incluso mediante este procedimiento, será necesaria la conversión del *String* que se desea escribir en un formato legible (*const char*) suponiendo el uso de Unicode una doble conversión ((LPCSTR)(LPCTSTR)).

```
int lengthLocalidad = LocalidadInfosalida.GetLength() * sizeof(TCHAR);

GeneracionArchivosSalida.write((LPCSTR)(LPCTSTR)LocalidadInfosalida,
lengthLocalidad);

GeneracionArchivosSalida << "\n";
```

- Las variables que son de tipo numérico, será factible su salida directa, utilizando el operador de escritura en archivo (o lectura según el caso).

```
GeneracionArchivosSalida << "Latitud zona instalación:" << "        " <<
LatitudInfosalida << "\n";

GeneracionArchivosSalida << "Angulo alfa óptimo:" << "             " <<
AlfaOptInfosalida << "\n";

GeneracionArchivosSalida << "Angulo beta óptimo:" << "             " <<
BetaOptInfosalida << "\n";

GeneracionArchivosSalida << "Parámetro K instalación:" << "         " <<
KInfosalida << "\n";
```

- Finalmente una vez insertados todos los datos en el archivo de texto generado, bastará con cerrarlo, y en su caso si el usuario lo requiere, abrirlo con el programa o aplicación que le pueda corresponder (bloc de notas, PDF…)

```
// cierre de archivo de salida
GeneracionArchivosSalida.close();

//muestra archivo salida
ShellExecute(NULL, _T("open"), _T("InformeSalida.txt"), NULL, NULL,
SW_SHOWNORMAL);
```

5.4.- Transmisión de información y eventos de ejecución

En el presente proyecto, al igual que en toda aplicación basada en cuadros de dialogo, es razonable pensar que el intercambio de información o entrada de datos generada por el usuario, sirve de base de cálculo, utilización o presenta una necesidad de procesado interna en el programa, siendo además que esta, estará disponible en la clase a la que pertenece el cuadro de dialogo en el que se ha implementado el intercambio de datos (como variable miembro del mismo), pero se requiere que la información esté disponible, para que se muestre, opere o manipule en otras clases diferentes de la de introducción del dato.

En este tipo de circunstancia, se presenta la problemática, de una variable miembro introducida en una determinada clase, no es legible ni mucho menos manipulable por ninguna otra clase del programa. Por tal motivo será necesaria que esta información, se pueda manipular en otras secciones o clases, circunstancia para la que se han empleado variables globales. Estas variables se implementan mediante la extensión *extern*, produciendo esto que la variable precedida de esta expresión, sea legible, accesible y manipulable en cualquier parte del programa en al que sea nombrada.

Por contrapartida este tipo de variable, no será asociable a ningún control o herramienta de la vista de programa, siendo así que se capturarán o mostrarán datos

mediante variables miembro, utilizando variables de intercambio, o auxiliares globales (las cuales se definirán de manera masiva en una archivo implementado específicamente para contenerlas y definirlas), siendo así accesibles a todos los que utilicen en su cabecera este archivo.

Así mismo y tal como se ha comentado en secciones anteriores, el programa avanza o se ejecuta en base a acciones. Cuando el usuario realiza una determinada acción (por ejemplo, pulsar un control de botón), se genera un mensaje, que hace una llamada a una acción, siendo que dentro de esta acción se implementará el código a ejecutar.

Los eventos que lanzan las acciones, se han definido en función del tipo de controles al que pertenecen o el flujo de desarrollo de la aplicación, habiéndose implementado por ejemplo:

- Evento que reacciona a un "clik" del botón izquierdo del ratón.

```
afx_msg void OnBnClickedOk();
```

- Selección de elementos de un control de lista

```
afx_msg void OnTcnSelchangeBasedatos(NMHDR *pNMHDR, LRESULT *pResult);
```

- Acciones ejecutadas a raíz de perdida s de foco sobre un elemento determinado.

```
afx_msg void OnEnKillfocusGdm();
```

- Eventos disparados por cambio de selección entre varios elementos.

```
afx_msg void OnCbnSelchangePr();
```

5.5.- Menús

La aplicación desarrollada, no solo está generada en una vista grafica de cuadros de dialogo, sino que a fin de obtener acceso a elementos generales de la aplicación, tales como manuales, pliegos de condiciones, o herramientas que puede ser de interés acceder en cualquier momento de la ejecución o desarrollo de un diseño de instalación, se ha dotado al software de menús para poder cubrir esta funcionalidad.

Tal desarrollo permite al usuario acceder a los controles o herramientas definidas en los menús, sin necesidad de encontrarse en ningún cuadro de dialogo en concreto, dado que las funciones son dependientes del propio menú desplegable y no de una clase concreta a la que pueda pertenecer cada vista de ejecución de programa.

La generación de un menú, corresponde, al igual que las vista de dialogo, en la generación del esqueleto de la misma mediante el propio editor de Visual C++,

desarrollando el esquema que se ha deseado tenga el menú, para nombrar o definir por un ID a cada uno de los elementos del menú.

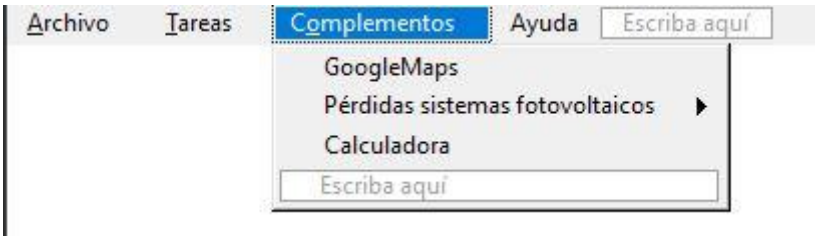


Figura 15: Menú desplegable aplicación

Una vez ejecutado el menú, al igual que con los controles o herramientas, se asociará cada uno de los capítulos o posibilidades de menú a un evento, que será invocado cuando se pulse dicho elemento (eventos definidos en la clase de cabecera o partida de la aplicación).

```
        afx_msg void OnCalcsombras();
        afx_msg void OnDiagorientacio();
        afx_msg void OnDiagsombras();
        afx_msg void OnGoogle();
        afx_msg void OnComplementosCalculadora();
        afx_msg void OnAppExit();
```

Estos eventos, desencadenarán una llamada a funciones, las cuales en su interior, albergarán el código necesario para ejecutar aquello a lo que hace referencia el menú, bien sea invocar un cuadro de dialogo modal, cerrar la aplicación, lanzar otras aplicaciones, etc…

```
void CPFCDlg::OnAyudaManualusuario()
{
        // TODO: Agregue aquí su código de controlador de comandos
        ShellExecute(NULL, _T("open"), _T("manual_usuario_Heclia.pdf"), NULL,
        NULL, SW_SHOWNORMAL);
}
```

5.6.- Seriación y generación de archivos

Una característica imprescindible de cualquier aplicación informática, es el poder guardar y recuperar datos de una sesión a otra. Los ficheros de datos permitirán almacenar la información que el usuario introduce. Un fichero de datos se almacena como un fichero separado en el disco duro, esto es, un fichero independiente del código de la aplicación.

Para poder generar y/o recuperar los archivos, se utilizarán las cajas de dialogo estándar del sistema Abrir y Guardar, mediante las cuales podremos escribir datos en un

fichero con un nombre determinado, o recuperar la información contenida en un fichero concreto.

Según esto, cuando el usuario desee guardar la información que ha ejecutado en la presente aplicación, se invocará a la caja de dialogo "*Guardar Como*", en la que el usuario escribirá un nombre de archivo (aquel con el que se desee guardar la información), siendo que se capturará el nombre escrito en la casilla correspondiente, asignándole la terminación de los archivos de la aplicación desarrollada (*.sfa*), generando el archivo y clasificándolo como de escritura.

```
void CPFCDlg::OnArchivoGuardar()
{
	this->UpdateData();
	CFileDialog DlgGuardarCambios(FALSE, NULL,_T(".sfa") ,
	OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT ,
	_TEXT("Ficheros de proyecto (*.sfa)|*.sfa |Ficheros de texto
	(*.txt)|*.txt| Todos los ficheros (*.*)|*.*||"), NULL);

	//visualiza caja de dialogo Guardar
	if (DlgGuardarCambios.DoModal() == IDOK)

	{
		//Visualizar el nombre del fichero en la caja de texto
		m_TextoEdit2 = DlgGuardarCambios.GetFileName();
		UpdateData(true);

		ExtraerTemporales(); //extrae datos de archivos temporales para
		seriar en archivo asociado

		//abrir el fichero para escribir
		CFile FicheroGuardado;
		//estructura para almacenar el estado del fichero
		CFileStatus EstadoFi;
		UINT nModoDeAceso = CFile::modeWrite;  //Acceso para escribir
```

El guardado en sí de archivos, se definirá como seriación, lo cual hace que los objetos que manipula la aplicación se hagan persistentes. Cuando guardemos los datos en un archivo, se abrirá el fichero que se ha generado con la caja de diálogo correspondiente, declarándolo como de escritura, e insertando en el archivo cada elemento, conjunto de elementos, u objetos que se necesiten, utilizando el operador "*ar*" que hace referencia al objeto *CArchive* ligado con la clase *CFile* que representa el fichero del disco, utilizándose en el caso del guardado el operador "<<" (al igual que cuando se generaban archivos de texto de salida), resultando que una vez escrito el archivo, bastará cerrarlo para que quede almacenado en un soporte físico.

```
//guardar datos generales aplicación
FicheroGuardado.Open(m_TextoEdit2, CFile::modeCreate | CFile::modeWrite);
CArchive ar(&FicheroGuardado, CArchive::store);

//sección DisSistemaRed
ar<< SeleccionInstalacion << LocalidadInfosalida<< ProvinciaInfosalida <<
PaisInfosalida << LatitudInfosalida << ComentariosInfosalida;

ar.Close();
//escribir el texto en el fichero
FicheroGuardado.Close();
```

De manera análoga al guardado de archivos, para la apertura de ficheros existentes, se realizará la misma maniobra, invocando al cuadro de dialogo "*abrir*", sobre el que se podrá seleccionar una archivo del disco duro del equipo, capturando el nombre del mismo y abriendo el fichero en modo lectura.

```
void CPFCDlg::OnBnClickedAbrirexistente()
{
	this->UpdateData();
	CFileDialog DlgAbrirExistente(TRUE, _T(".sfa"), NULL, OFN_HIDEREADONLY
	| OFN_OVERWRITEPROMPT,_TEXT("Ficheros de proyecto (*.sfa)|*.sfa
	|Ficheros de texto (*.txt)|*.txt| Todos los ficheros (*.*)|*.*||"),
	NULL);

	if (DlgAbrirExistente.DoModal()==IDOK)
	{
		//Visualizar el nombre del fichero en la caja de texto
		Registro = DlgAbrirExistente.GetFileName();
		UpdateData(true);
		RegistroAux = Registro;
```

La extracción de datos del interior del fichero, también se relacionará de manera íntima con el método empleado con el guardado, siendo que se abrirá archivo como lectura, se extraerá la información contenida mediante el operador inverso al guardado ">>", y posteriormente se cerrara el archivo manipulado.

```
//abrir el fichero para leer
CFile FicheroAbierto;
FicheroAbierto.Open(Registro, CFile::modeRead);//abro fichero

CArchive ar(&FicheroAbierto, CArchive::load);//abro fichero serializable

	ar >> SeleccionInstalacion;

	ar.Close();

FicheroAbierto.Close();//cierro ficheros
```

6.- Proyección/Evolución sistema desarrollado

El programa desarrollado, presenta un código abierto, el cual puede ser modificado en posteriores versiones a conveniencia del usuario o necesidades de cálculo que se consideren oportunas. Según esto no existirá restricción ninguna con respecto a posibles ampliaciones, pudiéndose, no solo mejorar elementos de código, sino pantallas y funcionalidades completas para aumentar el rango de actuación o posibilidades de cálculo, gestión y/o apoyo al usuario en el planeamiento de instalaciones renovables.

Como ejemplo de algunas posibilidades o funciones de uso o ampliación del sistema se pueden mencionar:

- Activación y desarrollo de funciones encaminas al cálculo de instalaciones renovales mediante aerogeneradores, o instalaciones mixtas fotovoltaicas-eólicas, las cuales se prevén en el presente proyecto, pero no se ejecutan.

- Ejecución de otro tipo de energías alternativas de aplicación en sectores agropecuarios, como puede ser la energía solar térmica (calentamiento agua), o la utilización de biogás (granjas porcinas, etc…).

- Realización de bases de datos o instalaciones precalculadas de consumo agropecuario (granjas avícolas, porcinas, instalaciones agrícolas, etc…), en las que sobre una base de desarrollo sean introducibles particularidades sobre puntos de consumo estándares.

- Enlace de base de datos de materiales y elementos (con aplicaciones en función de nuevos tipos de instalaciones tipo eólico u otros), que permitan seleccionar los elementos a instalar en el cálculo directamente desde la base de datos e integrar sus propiedades en el resto del sistema de cálculo de la instalación.

- Incremento de capacidades de cálculo o diseño, con posible ejecución de planimetrías, modelos visuales, e inserción de fotografías para trabajo sobre las mismas.

- Realización de cálculos estructurales, de sistemas de instalaciones tipo seguidores y gestión global de dichos cálculos (calculo tensiones, deformaciones, acciones, etc…)

- Calculo y gestión global del proyecto, incluyendo cálculo de líneas de distribución, transformadores, protecciones, contadores, y en general cualquier elemento asociados a la instalación, incluyéndose en su caso, tal como se han citado en apartados previos, planimetría, esquemas, etc…

7.- Bibliografía

IDAE. “Pliego de Condiciones Técnicas de Instalaciones Conectadas a Red”. Instituto para la Diversificación y Ahorro de la Energía, julio 2011. [1]

IDAE. “Pliego de Condiciones Técnicas de Instalaciones Aisladas de Red”. Instituto para la Diversificación y Ahorro de la Energía, febrero 2009. [2]

Jon Bates & Tim Tompkins. “Microsoft Visual C++ 6”. Prentice Hall Iberia S.R.L., 1999, ISBN: 84-8322-077-6. [3]

Francisco Javier Ceballos. “Visual C++, aplicaciones para Win32”. Ra-Ma Editorial, febrero 1999, 2º edición. ISBN: 84-7897-350-8. [4]

Beck Zaratian. “Microsoft Visual C++ 6.0, Manual del Programador”. McGraw-Hill, 1998. ISBN: 84-481-2127-9. [5]

Francisco Javier Ceballos. “Programación orientada a objetos con C++”. Ra-Ma Editorial, Septiembre 1997, 2º edición. ISBN: 84-7897-268-4. [6]

Ministerio de Fomento. “Código Técnico de la Edificación”. Real Decreto 314/2006 de 17 de marzo. [7]