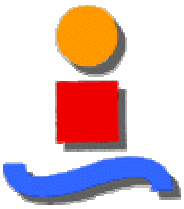


EOCEN: Sistema de Control de la Calidad de la Energía Eólica

Memoria de Cálculo

Gonzalo Pérez Noguerol

Tutor: D. José Luis Calvo Borrego

Departamento de Ingeniería Electrónica
Universidad de Sevilla

Junio de 2006

1. Introducción

El presente documento recoge de manera detallada los distintos pasos llevados a cabo para implementar los módulos que componen el sistema de control central de parque del proyecto EOCEN, cumpliendo con los requisitos planteados en la memoria descriptiva y justificativa de este proyecto.

Así mismo, describe el proceso seguido para integrar todos los bloques y poner en marcha el sistema prototipo, que tiene como objetivo probar las bondades de EOCEN en campo, instalándolo en dos aerogeneradores del parque eólico de Tahivilla, actualmente explotado por Endesa, empresa que será la encargada de verificar los resultados del sistema.

Además, se presentarán dos anexos con las directrices marcadas por el Real Decreto 436/2004 sobre la producción de la energía eólica y los mecanismos impuestos por la norma CEI 61000-3-6 (Anexo 2) para la realización de los estudios de calidad de la energía producida.

2. Implementación de la base de datos

El almacenamiento de los datos recogidos se realizará haciendo uso de un SGBD, PostgreSQL. Postgre es una potente base de datos relacional de fuentes abiertas optimizada para el tratamiento de grandes volúmenes de información, garantizando la integridad referencial y permitiendo la implementación de rutinas en PL-SQL.

En este apartado se recoge el proceso llevado a cabo para instalar y configurar la base de datos PostgreSQL en el servidor Linux del sistema EOCEN, crear la estructuración física e implementar las distintas estructuras lógicas para el almacenamiento de los datos.

2.1. Instalación y configuración de la BBDD PostgreSQL 8.0.4

El primer paso para instalar la base de datos en el servidor Linux, con sistema operativo Ubuntu, consistió en descargar la versión deseada, en este caso la 8.0.4, del código fuente desde la página del proyecto Postgre (http://www.postgresql.org/download/), asegurándonos que dicha distribución es la adecuada para el sistema operativo del servidor sobre el que va a funcionar.

El fichero descargado, *postgresql-8.0.4.tar.bz2,* estaba comprimido, por lo que necesitamos utilizar el descompresor *tar* para obtener de él los paquetes necesarios para la instalación. El comando utilizado fue:

```
tar – xvjf postgresql-8.0.4.tar.bz2
```

El siguiente paso (de aquí en adelante, todo el proceso fue hecho con el usuario *root*), fue compilar el código fuente mediante la ejecución del script *start.* Durante la ejecución del mismo, se testea si el sistema operativo tiene instaladas y configuradas las librerías necesarias para el funcionamiento de la Postgre. Dicha comprobación dio como resultado que faltaban las siguientes librerías:

- gcc
- libreadline5-dev
- libc6-dev
- zlib1g-dev

Los comandos utilizados para instalar dichas librerías y posteriormente compilar el código fuente de la base de datos fueron:

```
apt –get install gcc libc6-dev libreadline5-dev zlib1g-dev make
./start
make install
```

Este último comando es el encargado de, una vez compilado el código fuente, instalar el servidor Postgre en el directorio */usr/local/pgsql*.

Una vez finalizada la instalación, se procedió a configurar el servicio Postgre para obtener el funcionamiento adecuado. Los pasos seguidos fueron [7]:

- Añadir la ruta del ejecutable al path: en el archivo */etc/profile* se incluyó la ubicación */usr/local/pgsql/bin*
- Crear un usuario *postgres* bajo el cual funcionará la base de datos. Para ello:
  - Se insertó en el archivo de configuración de grupos */etc/group* la línea *postgres:x:1001:postgres* (añade un grupo postgres con GID 1001 y con un único usuario llamado *postgres*)
  - Se creó la carpeta */var/lib/postgres* y se modificó el dueño y permisos de este directorio para que el usuario *postgres* tuviera dominio total sobre el mismo:

```
mkdir /var/lib/postgres
chown postgres /var/lib/postgres
chmod 770 /var/lib/postgres
```

  - Se asignó dicho directorio como home del usuario *postgres* mediante la inclusión en el archivo */etc/passwd* de la línea de configuración *postgres:x:1001:1001::var/lib/postgres*.
  - Se asigno “postgres” como contraseña del usuario *postgres*:

```
password postgres postgres
```

- Inicializar la base de datos

```
initdB –D /var/lib/postgres
```

- Crear el directorio */etc/postgres* (perteneciente al usuario *postgres*) en el que debe haber enlaces simbólicos a los archivos de configuración *pg_hba.conf, postgresql.conf* y *pg_ident.conf* y una carpeta para guardar los log de la Postgre:

```
mkdir /etc/postgres
mkdir /etc/postgres/log

chown –R postgres /etc/postgres
chmod –r 770 /etc/postgres

ln –s /var/lib/postgres/pg_hba.conf  /etc/postgres/pg_hba.conf
ln –s /var/lib/postgres/ postgresql.conf/ /etc/postgres/postgresql.conf
ln –s /var/lib/postgres/ pg_ident.conf /etc/postgres/ pg_ident.conf
```

- Revisar las rutas y parámetros del script de arranque */etc/init.d/postgres*

- Modificar los ficheros de configuración para permitir la conexión y accesos a las base de datos:

  - Archivo *pg_hgba.conf* : Contiene una lista de las IP de los equipos a los que se les permite conectarse a cada base de datos, así como el método de autenticación. En él se añadió, en el apartado reservado para conexiones por IPv4, la línea necesaria para permitir a todos los equipos de la subred del parque, con IP 192.168.x.x, la conexión a la base de datos eocen a través del usuario *eocen* :

```
host    eocen    eocen    192.168.0.0/16    password
```

  - Archivo *postgresql.conf*: En el se introducen los parámetros de configuración que rigen el funcionamiento de la base de datos. Para garantizar que el servicio atiende las peticiones de conexión por el puerto adecuado, se incluyeron las siguientes líneas:

```
listen_addresses = '*'  # what IP interface(s) to listen on;
port = 5432
```

2.2. Creación de las estructuras físicas y lógicas para el almacenamiento de datos

Para poder generar la base de datos y su organización interna, se utilizó la herramienta pgAdmin, que permite administrar remotamente, a través de una interfaz gráfica, cualquier base de datos PostgreSQL, evitando tener que trabajar directamente sobre la línea de comandaos del servidor.

Tras conectarnos al gestor de la Postgre, se siguieron los siguientes pasos:

- Crear un usuario *eocen* con password *eocen.* Este será el futuro dueño de la base de datos, y cualquier conexión a la misma deberá hacerse logándose con dicha contraseña.

- Crear la base de datos *eocen*, perteneciente al usuario *eocen*, y con formato de codificación UTF8, que permite almacenar datos de 8 bits.

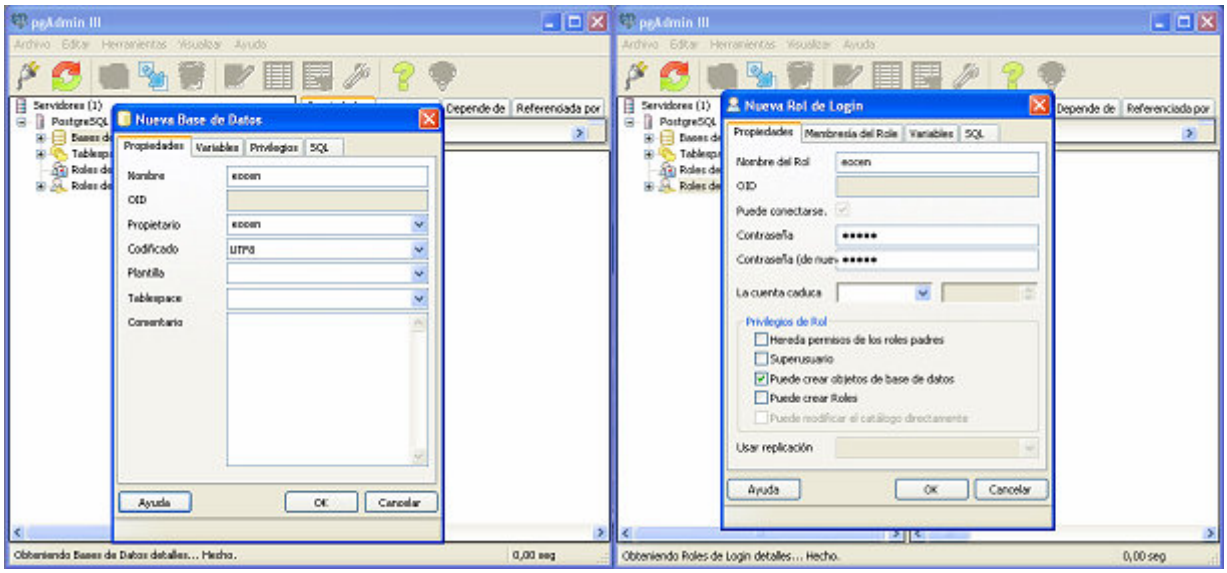


Imagen 1: Creación del usuario y de la base de datos

- Crear los 3 tablespaces en los que se organizará la estructura física de la base de datos: *eocen_estat*, para tablas de datos de sólo lectura; *eocen_lect*, para tablas de datos preferentemente de lectura y con un número de inserciones bajo; y *eocen_escrit*, para tablas de datos con un número elevado de inserciones y reducido de accesos de lectura. Aunque en la memoria justificativa se indicó que cada uno de los tablespaces tendría asignado un disco duro diferente para optimizar el rendimiento de la base de datos y garantizar la escalabilidad del sistema, el servidor del prototipo experimental consta de un único disco duro, por lo que todos los tablespaces estarán recogidos en el mismo en distintas carpetas en el directorio */var/pgsql/*. El propietario de todos ellos es el usuario *eocen*.

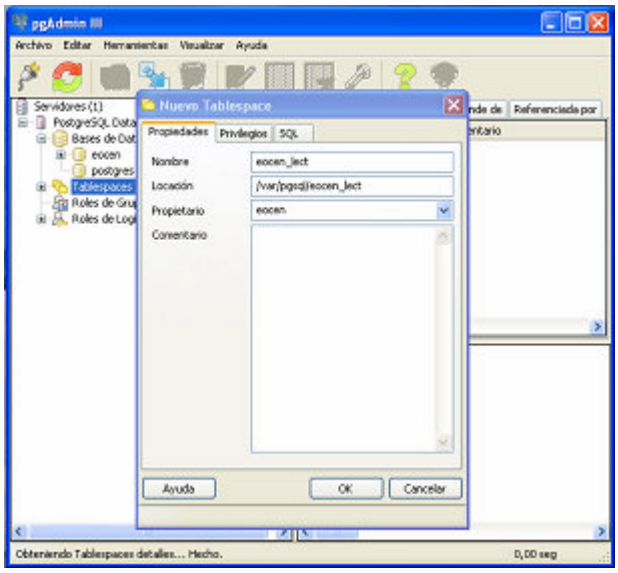
- Creación de los 5 esquemas (propiedad todos ellos del usuario *eocen*) en los que se organizarán las distintas tablas de la base de datos:

  - **eocen_conf**: Esquema que alberga las tablas de configuración estática de todas las aplicaciones del sistema EOCEN.

  - **eocen_arp**: Esquema que contiene las tablas para almacenar las tramas de datos de los ARP y los ADS, así como los resúmenes de producción.

  - **eocen_calidad**: Esquema que alberga las tablas para almacenar los estudios de calidad, tanto los recibidos de 3 segundos como los calculados de 10 minutos.

  - **eocen_cyclics**: Esquema que alberga las tablas para almacenar los cyclics recibidos tanto por evento como por petición.



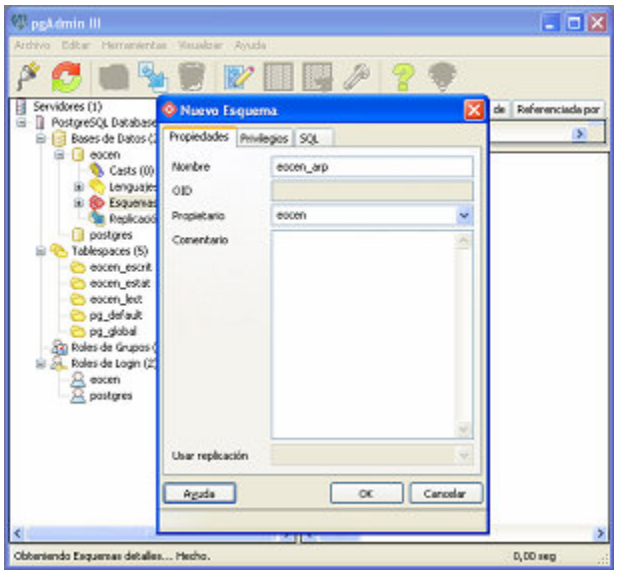


Imagen 2: Creación de los tablespaces y esquemas

- Creación de la estructura de tablas necesaria para albergar los datos. Para permitir regenerar la base de datos o incluir modificaciones en la misma, en caso que fuera necesario, minimizando el coste temporal, se escribieron unos scripts encargados de generar las distintas tablas. De esta forma, basta con ejecutar dichos scripts para obtener de una forma rápida y sencilla todas las tablas con la información estática (aquella que hace referencia a la configuración de los distintos módulos) ya insertada.

- Para permitir la escalabilidad de la base de datos y permitir alcanzar los niveles de jerarquía planteados (parque, nudo, región y sistema), se definió una nomenclatura ortogonal para las variables a almacenar, cuya estructura es:

Sistema_Region_Nudo_Parque_Máquina_Tipo_Variable

  - Sistema: un carácter indicativo del país. En el caso del prototipo será “E” (España)

  - Región: permitirá distinguir la región a la que pertenece el parque que contiene la máquina cuyos datos vamos a recogerán. Este campo estará formado por 2 caracteres identificativos. Para el prototipo será “AN” (Andalucía)

  - Nudo: 2 caracteres que identificaran el nudo dentro de la región. Si los datos son de una región en vez de un aerogenerador, este campo valdrá '00'. Para el sistema prototipo será “TV” (Tahivilla)

  - Parque: campo formado por 2 caracteres que permitirá identificar el parque que contiene al aerogenerador cuyos datos vamos a almacenar. En el caso de datos de todos los parques de un nudo, este campo valdrá '00'. Para el prototipo experimental será “TV”, Tahivilla

  - Máquina: campo formado por 3 dígitos que identifica la máquina (empezando por la “001”). Para datos de todos los analizadores de un parque utilizaremos el identificador “0002

  - Tipo: campo de 3 caracteres que indicará el tipo de mensaje del que se han recogido los datos. Estos tipos de mensaje son:

Tipo	Tipo de Mensaje
ARP	Datos de ARP (mensajes ARP y ARP + ADS)
ADx	Datos del ADSx, donde x puede ser 1 o 2 (mensajes ARP + ADS)
CYE	Cyclic generado por evento programable
CYP	Cyclyc generado por petición de usuario
QOS	Estudio de calidad
CAL	Variables calculadas
CNF	Mensaje de configuración de ARP
PET	Mensaje de petición al ARP

Tabla 1: Posibles valores para el campo “TIPO”

- Variable: Estará formado por 3 subcampos de 2, 3 y 4 caracteres respectivamente, separados por un guión bajo que indicarán la variable dentro del mensaje. Estos subcampos indicarán:
  - 1º: tipo de variable: diezminutal (XM), secundal (XS), general (XX), configuración (CF), petición (RQ) o control (CT)
  - 2º y 3º: variable y si es necesario, unidades.

2.2.1. Tablas de eocen_conf

En este esquema se encuentran las tablas necesarias para definir las constantes necesarias para el funcionamiento de las aplicaciones así como la configuración particular de la instalación. Todos estos datos son estáticos, por lo que las tablas se asignan al tablespace *eocen_estat*, salvo en el caso de *conf_recogidas, conf_recogidas_log y conf_configuraciones*, cuyo contenido puede variar dinámicamente, por lo que pertenecerán al tablespace *eocen_lect*.

- **Tablas de configuración del módulo de cálculo:** *calc_operandos, calc_puertascalculadas* y *def_calculos***.** Contienen las variables a calcular, las puertas que se utilizan como operando en cada uno de los cálculos, el tipo de operación a realizar y el orden de las mismas.
- **Tablas de configuración del módulo de bombeo:** *conf_bombeo_inm, conf_bombeo_calidad_a* y *conf_bombeo_calidad_i.* Contienen, para cada tipo de

variable, la periodicidad con la que el módulo de bombeo debe hacer las inserciones de la misma en la base de datos.

- **Tablas de configuración del módulo de alarmas:** *def_alarmas y def_alarmas_tipo,* y *conf_alarmas_patrones* Contienen los tipos de alarmas que el sistema es capaz de lanzar así como el tipo de datos que se recogen cada vez que se produce una y los parámetros de configuración del módulo de alarmas. Se incluyó además una tabla *conf_configuraciones*, con la configuración de los eventos programados para cada ARP.

- **Tablas para la configuración del driver de comunicaciones:** *conf_puertas* relaciona los identificadores de cada puerta con su nomenclatura; *conf_recogidas***,** almacena los periodos de recogida de datos para cada ARP (los campos indican para cada ARP cada cuánto tiempo debe el driver de comunicaciones solicitarle los datos y si se incluyen datos del ADS correspondiente). Hay además un trigger sobre esta tabla cuya misión es que cada vez que se actualice una fila en ella, meter una fila en *conf_recogidas_log*, tabla que tiene todos los campos de *conf_recogidas* mas uno adicional "fecha_hora" y que forma un histórico automático de los cambios en *conf_recogidas*. En último lugar, se incluyó una tabla *def_mensajes* con las definiciones de los tipos de mensajes existentes para el driver de comunicaciones.

- **Tablas con la estructura jerárquica del sistema:** *conf_sistemas, conf_regiones, conf_nodos, conf_parques, conf_subparques* y *conf_aeros*. Recogen la estructura organizativa del sistema: aerogeneradores que forman cada subparque, subparques en que se organiza un parque, parques que contiene cada nodo, nodos de cada región y regiones de cada sistema.

- **conf_arps**: ARP's registrados en el sistema, con su IP, modelo y parque asociado. La identificación de cada ARP es "*PP_NNN*" con *PP* siendo el parque y *NNN* el número del aerogenerador en el que se instala.

- **conf_adss**: ADS's registrados en el sistema. Cada ADS está asociado a un ARP y en su nomenclatura incluye un número identificativo al final, quedando "*PP_NNN_D*" siendo *D* el número de ADS.

- **conf_parametros**: Esta tabla almacena parámetros puntuales (parejas clave-valor) para cualquier aplicación que los necesite.

- **def_cei61000-3-6**: Valores definidos por la norma CEI 61000-3-6 para los armónicos.

- **def_medidas**: Definiciones de los parámetros eléctricos del ARP acompañadas de sus correspondientes unidades.

- **def_schemas**: contiene la equivalencia entre los campos de la nomenclatura de las puertas y su significado.

Además de todas estas tablas, se definieron 7 vistas sobre ellas que son a las que realmente acceden los módulos de cálculo y bombeo para consultar sus parámetros de configuración: *conf_alarmas, conf_bombeo, conf_bombeo_calidad_a_view, conf_bombeo_calidad_i_view, conf_bombeo_patrones y conf_puertascompletas.*

Los scripts necesarios para generar todas estas tablas y su contenido se han omitido debido a motivos de confidencialidad del proyecto.

2.2.2. Tablas de eocen_arp

Este esquema contiene todas las tablas necesarias para almacenar los datos sobre los distintos parámetros eléctricos enviados por los ARP y ADS del sistema.

Para ello, se definieron dos estructuras de tabla padre, una con los campos necesarios para albergar los datos enviados por los ARP y otra para los de los ADS y se generó una tabla hija que heredase de cada una de ellas por cada aerogenerador. Las tablas están indexadas por la fecha y hora de los datos recogidos en la fila. Todas estas tablas pertenecen al tablespace *eocen_escrit.*

- **Tabla padre para los datos de los ARP**: el script para generar dicha tabla es:

```
-- Table: eocen_arp.arp_type

CREATE TABLE eocen_arp.arp_type
(
  fecha_hora timestamptz NOT NULL,   -- fecha y hora de los datos
  vfr_int2 int2[],                   -- array con la forma de onda de Vr
  vfs_int2 int2[],                   -- array con la forma de onda de Vs
  vft_int2 int2[],                   -- array con la forma de onda de Vt
  ifu_int2 int2[],                   -- array con la forma de onda de Iu
  ifv_int2 int2[],                   -- array con la forma de onda de Iv
  ifw_int2 int2[],                   -- array con la forma de onda de Iw
  vfr_efft int2[],                   -- array con el espectro de Vr
  vfs_efft int2[],                   -- array con el espectro de Vs
```

```
vft_efft int2[],          -- array con el espectro de Vt
ifu_efft int2[],          -- array con el espectro de Iu
ifv_efft int2[],          -- array con el espectro de Iv
ifw_efft int2[],          -- array con el espectro de Iw
vfr_vrms float4,          -- valor eficaz de Vr
vfs_vrms float4,          -- valor eficaz de Vs
vft_vrms float4,          -- valor eficaz de Vt
ifu_vrms float4,          -- valor eficaz de Iu
ifv_vrms float4,          -- valor eficaz de Iv
ifw_vrms float4,          -- valor eficaz de Iw
pfr_acge float4,          -- valor de  pot. activa generada fase R
pfs_acge float4,          -- valor de  pot. activa generada fase S
pft_acge float4,          -- valor de  pot. activa generada fase T
pfr_acco float4,          -- valor de  pot. activa consumida fase R
pfs_acco float4,          -- valor de  pot. activa consumida fase S
pft_acco float4,          -- valor de  pot. activa consumida fase T
pfr_rein float4,          -- valor de  pot. reactiva inductiva fase R
pfs_rein float4,          -- valor de  pot. reactiva inductiva fase S
pft_rein float4,          -- valor de  pot. reactiva inductiva fase T
pfr_reca float4,          -- valor de  pot. reactiva capacitiva fase R
pfs_reca float4,          -- valor de  pot. reactiva capacitiva fase S
pft_reca float4,          -- valor de  pot. reactiva capacitiva fase T
pfr_apar float4,          -- valor de  pot. aparente fase R
pfs_apar float4,          -- valor de  pot. aparente fase S
pft_apar float4,          -- valor de  pot. aparente fase T
pfr_fact float4,          -- factor de potencia fase R
pfs_fact float4,          -- factor de potencia fase S
pft_fact float4,          -- factor de potencia fase T
vfr_xthd float4,          -- factor de distorsión armónica de Vr
vfs_xthd float4,          -- factor de distorsión armónica de Vs
vft_xthd float4,          -- factor de distorsión armónica de Vt
ifu_xthd float4,          -- factor de distorsión armónica de Iu
ifv_xthd float4,          -- factor de distorsión armónica de Iv
ifw_xthd float4,          -- factor de distorsión armónica de Iw
vfn_vrms float4,          -- valor eficaz de tensión fase - neutro
i3f_vrms float4,          -- valor eficaz de intensidad trifásica
v3f_vrms float4,          -- valor eficaz de tensión trifásica
p3f_acge float4,          -- valor de pot. trifásica activa generada
p3f_acco float4,          -- valor de pot. trifásica activa consumida
p3f_rein float4,          -- valor de pot. trifásica reactiva inductiva
p3f_reca float4,          -- valor de pot. trifásica reactiva capacitiva
p3f_apar float4,          -- valor de pot. trifásica aparente
p3f_fact float4,          -- factor de potencia trifásica
efr_acge float4,          -- valor de energía activa generada fase R
efs_acge float4,          -- valor de energía activa generada fase S
eft_acge float4,          -- valor de energía activa generada fase T
efr_acco float4,          -- valor de energía activa consumida fase R
efs_acco float4,          -- valor de energía activa consumida fase S
eft_acco float4,          -- valor de energía activa consumida fase T
efr_rein float4,          -- valor de energía reactiva inductiva fase R
efs_rein float4,          -- valor de energía reactiva inductiva fase S
eft_rein float4,          -- valor de energía reactiva inductiva fase T
efr_reca float4,          -- valor de energía reactiva capacitiva fase R
efs_reca float4,          -- valor de energía reactiva capacitiva fase S
eft_reca float4,          -- valor de energía reactiva capacitiva fase T
e3f_acge float4,          -- valor de energía trifásica activa generada
e3f_acco float4,          -- valor de energía trifásica activa consumida
e3f_rein float4,          -- valor de energía trifásica reactiva inductiva
e3f_reca float4,          -- valor de energía trifásica reactiva capacitiva
```

```
  x3f_frec float4,                -- valor de frecuencia trifásica
  vfr_fesc float4,                -- factor de escala de los datos de vfr_int2
  vfs_fesc float4,                -- factor de escala de los datos de vfs_int2
  vft_fesc float4                 -- factor de escala de los datos de vft_int2,
  ifu_fesc float4,                -- factor de escala de los datos de ifu_int2
  ifv_fesc float4,                -- factor de escala de los datos de ifv_int2
  ifw_fesc float4,                -- factor de escala de los datos de ifw_int2
  vrs_vrms float4,                -- valor eficaz de la tensión Vrs
  vst_vrms float4,                -- valor eficaz de la tensión Vst
  vtr_vrms float4,                -- valor eficaz de la tensión Vtr
  u32_fech int4                   -- segundos desde 00:00h de 1/1/1980
)
WITHOUT OIDS TABLESPACE eocen_escrit;
ALTER TABLE eocen_arp.arp_type OWNER TO eocen;
```

Como se observa, salvo el campo fecha_hora, utilizado como índice de la tabla, no hay restricciones sobre los valores de los datos (pueden ser nulos), ya que por diversos motivos, el driver puede decidir que alguno de los datos que recibe en la trama no es válido.

El prototipo elaborado para la fase experimental consta sólo de 2 ARP instalados en los aerogeneradores 002 y 412, por lo que se implementaron 2 tablas que heredasen de *eocen_arp_type* con los nombres *tv_xxx_arp*, dónde “xxx” es el número del aerogenerador donde está instalado el ARP y “tv” son las siglas identificativas del parque eólico de Tahivilla.

El script para generar las tablas que heredan de *eocen_arp_type* es:

```
-- Table: eocen_arp.tv_xxx_arp

CREATE TABLE eocen_arp.tv_xxx_arp
(
  CONSTRAINT tv_xxx_arp_pkey PRIMARY KEY (fecha_hora)
) INHERITS (eocen_arp.arp_type)
WITHOUT OIDS TABLESPACE eocen_escrit;
ALTER TABLE eocen_arp.tv_xxx_arp OWNER TO eocen;
```

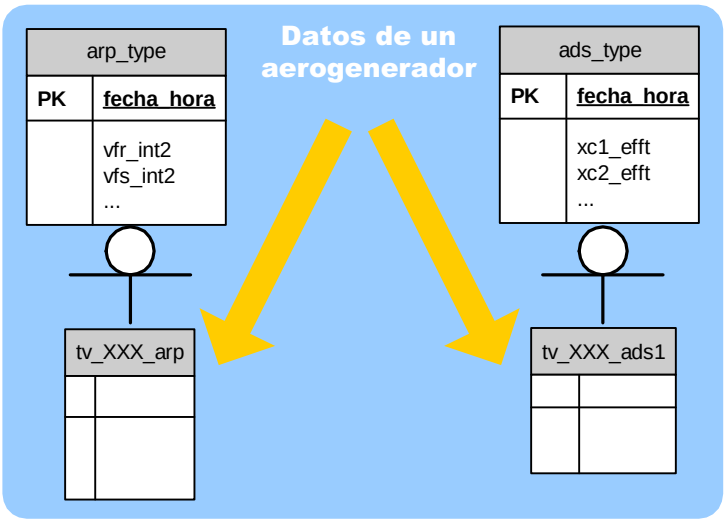


Imagen 3: Estructura para almacenar los datos de un aerogenerador

- **Tabla padre para los datos de los ADS:** el script para generar dicha tabla es:

```
-- Table: eocen_arp.ads_type

CREATE TABLE eocen_arp.ads_type
(
  fecha_hora timestamptz NOT NULL,     -- fecha de los datos
  xc1_efft int2[] NOT NULL,            -- array con el espectro del canal 1
  xc2_efft int2[] NOT NULL,            -- array con el espectro del canal 2
  xc3_efft int2[] NOT NULL,            -- array con el espectro del canal 3
  xc4_efft int2[] NOT NULL,            -- array con el espectro del canal 4
  xc1_vrms float4 NOT NULL,            -- valor eficaz  del canal 1
  xc2_vrms float4 NOT NULL,            -- valor eficaz  del canal 2
  xc3_vrms float4 NOT NULL,             -- valor eficaz  del canal 3
  xc4_vrms float4 NOT NULL             --  valor eficaz  del canal 4
)
WITHOUT OIDS TABLESPACE eocen_escrit;
ALTER TABLE eocen_arp.ads_type OWNER TO eocen;
```

De nuevo nos encontramos con que el prototipo experimental sólo posee 2 ADS, uno asociado a cada ARP, por lo que sólo hubo que generar 2 tablas hijas con la nomenclatura *tv_xxx_adsz*, donde “tv” es el identificador del parque eólico de Tahivilla, “xxx” el número del aerogenerador en el que está instalado el ADS, y la “z” el número del ADS del aerogenerador (recordemos que cada aerogenerador pude tener varios ADS asociados).

El script para generar las tablas hijas es:

```
-- Table: eocen_arp.tv_xxx_ads1

CREATE TABLE eocen_arp.tv_xxx_ads1
(
  CONSTRAINT tv_xxx_ads1_pkey PRIMARY KEY (fecha_hora)
) INHERITS (eocen_arp.ads_type)
WITHOUT OIDS TABLESPACE eocen_escrit;
ALTER TABLE eocen_arp.tv_xxx_ads1 OWNER TO eocen;
```

2.2.3. Tablas de eocen_alarmas

Este esquema contiene todas las tablas necesarias para almacenar los datos referentes a las alarmas que se produzcan por la programación de eventos así como los cyclics producidos por una alarma o por la petición por parte del operador. Por ser tablas destinadas a la inserción continua de datos, todas ellas se asignaron al tablespace *eocen_escrit*.

- **Tabla tv_alarmas**: almacena un histórico de las alarmas que se han producido en el sistema, sus fechas de inicio y fin, la causa y la zona en la que se han producido:

```
-- Table: eocen_alarmas.tv_alarmas

CREATE TABLE eocen_alarmas.tv_alarmas
(
  id_puerta int4 NOT NULL,                  -- identificador de la puerta
  fecha_ini timestamptz NOT NULL,           -- fecha de inicio de alarma
  fecha_fin timestamptz,                    -- fecha de fin de la alarma
  id_alarma int4 NOT NULL,                  -- identificador de la alarma
  descripcion varchar(100) NOT NULL,        -- descripción de la alarma
  severidad int4 NOT NULL,                  -- severidad de la alarma
  zona int4 NOT NULL,                       -- zona en la que se produce

  CONSTRAINT tv_alarmas_pk PRIMARY KEY (id_puerta, fecha_ini, id_alarma),

  CONSTRAINT tv_alarmas_fk1 FOREIGN KEY (id_alarma)
     REFERENCES eocen_conf.def_alarmas (id_alarma) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE CASCADE,

 CONSTRAINT tv_alarmas_fk2 FOREIGN KEY (id_puerta)
     REFERENCES eocen_conf.conf_puertas (id_puerta) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE CASCADE
)
WITHOUT OIDS;
ALTER TABLE eocen_alarmas.tv_alarmas OWNER TO eocen;
```

- **Tablas para almacenar los cyclics**: Cada cyclic está compuesto por una campo *fecha_hora* más un vector que comienza con la causa de la alarma y el número de muestras recogidas, junto con sus factores de escala, y en el que a continuación se intercalan alternativamente las muestras de las 3 intensidades y las 3 tensiones trifásicas recogidas durante el instante en que se produce la alarma y los instantes posteriores. A la hora de procesar todos estos datos, podemos optar por realizar la descomposición antes de almacenarlos o introducirlos en la base de datos tal cual llegan y realizar la descomposición cuando sea necesario.

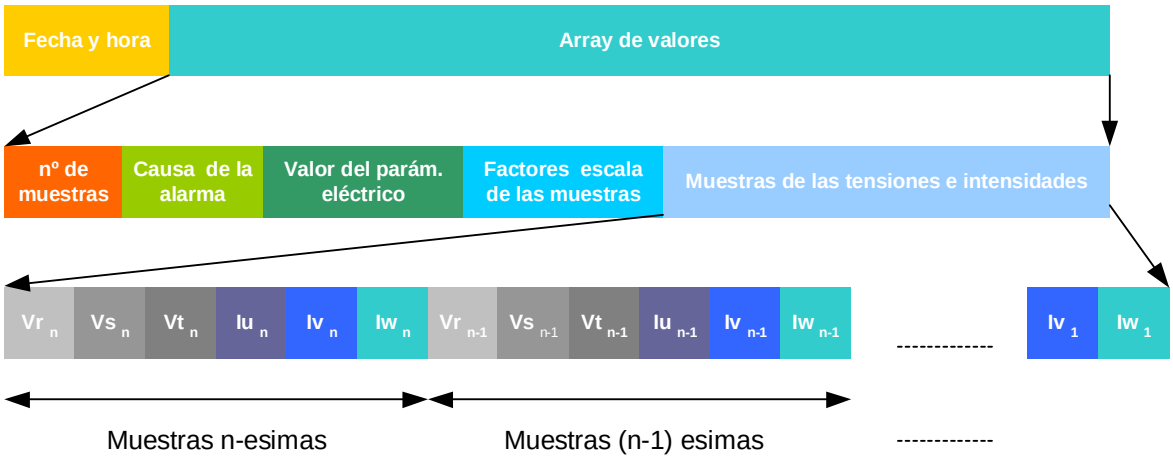


Imagen 4: Composición de un cyclic

Debido al enorme coste temporal de separar los datos (la operación puede llegar a durar más de un minuto), se adoptó la primera solución, que aunque supone un retraso desde que se reciben los datos hasta que pasan a estar disponibles, permite que el acceso a los mismos a través de la interfaz de control una vez que están disponibles sea inmediato, evitando esperas al operador.

Para hacer esto posible, se implementó, usando de nuevo la herencia, una tabla *cyc_type* por ARP formada por los campos *fecha_hora*, *id_puerta* y un array de valores en la que se introducen los datos tal cual llegan y otra tabla *cyc_type_desc* en la que se almacenan ya descompuestos, junto con un procedimiento PL/SQL encargado de la descomposición que se activa cada vez que hacemos una inserción en la primera tabla. Para evitar la duplicidad de los datos, el procedimiento, una vez que acaba de procesar los datos, borra la entrada de la tabla *cyc_type.*

El código SQL de las 2 clases padre es el siguiente:

```
-- Table: eocen_alarmas.cyc_type

CREATE TABLE eocen_alarmas.cyc_type
(
  fecha_hora timestamptz NOT NULL,          -- fecha de los datos
  id_puerta int4 NOT NULL,                  -- identificador de puerta
  valor float8[] NOT NULL                   -- array con los datos
)
WITHOUT OIDS TABLESPACE eocen_escrit;
ALTER TABLE eocen_alarmas.cyc_type OWNER TO eocen;
```

```
-- Table: eocen_alarmas.cyc_desc_type

CREATE TABLE eocen_alarmas.cyc_desc_type
(
  fecha_hora timestamp NOT NULL,            -- fecha de los datos
  causa int2 NOT NULL,                      -- causa de la alarma
  valor int2 NOT NULL,                      -- valor del parámetro eléctrico
  fvr float8 NOT NULL,                      -- factor de escala de Vr
  fvs float8 NOT NULL,                      -- factor de escala de Vs
  fvt float8 NOT NULL,                      -- factor de escala de Vt
  fiu float8 NOT NULL,                      -- factor de escala de Iu
  fiv float8 NOT NULL,                      -- factor de escala de Iv
  fiw float8 NOT NULL,                      -- factor de escala de Iw
  vfr_int2 int2[] NOT NULL,                 -- forma de onda de Vr
  vfs_int2 int2[] NOT NULL,                 -- forma de onda de Vs
  vft_int2 int2[] NOT NULL,                 -- forma de onda de Vt
  ifu_int2 int2[] NOT NULL,                 -- forma de onda de Iu
  ifv_int2 int2[] NOT NULL,                 -- forma de onda de Iv
  ifw_int2 int2[] NOT NULL                  -- forma de onda de Iw
)
WITHOUT OIDS;
ALTER TABLE eocen_alarmas.cyc_desc_type OWNER TO eocen;
```

El código PL/SQL encargado de la descomposición de los datos es el siguiente:

```
-- Function: eocen_alarmas.cyc_view2(tablename "varchar", vector _float8, fecha_hora timestamptz)

CREATE OR REPLACE FUNCTION eocen_alarmas.cyc_view2(tablename "varchar", vector _float8,
fecha_hora timestamptz)
  RETURNS void AS
$BODY$DECLARE

  --constantes y variables
  BASEPOS int2 := (9+1);          -- Indica donde empiezan los datos de las ondas
  SIZEPOS int2 := 1;              -- Posicion del array donde esta numero de muestras que contiene el ciclyc
  NUMPARAM int2 := 6;             -- Numero de arrays en que se descompone el array recogido
  NUMPARAM1 int2 := NUMPARAM-1; -- para indexar los arrays
```

```
 tamTrama int2;                    -- numero de muestras que contiene el array
 aux int4;
 tabla varchar;

 array_gen varchar[]:='{}';        -- tabla de string para almacenar los resultados

BEGIN

    -- inicializo los 6 arrays(string) que contendrán las formas de onda de las tensiones e intensidades
    FOR j in 0..NUMPARAM1 LOOP
       array_gen[j]:='{';
    END LOOP;

    -- calculo el numero de muestras
    tamTrama:= vector[SIZEPOS];

    -- recorro todas las muestras
    FOR i IN 0..tamTrama LOOP

                        aux := BASEPOS+(i*NUMPARAM);

                         -- para cada muestra, saco los 6 puntos
                        FOR j IN 0..NUMPARAM1 LOOP
                         array_gen[j]:= array_gen[j] || vector[aux+j] || ',';
                END LOOP;

    END LOOP;

    -- pongo los 6 arrays bonitos
    FOR j in 0..NUMPARAM1 LOOP

                array_gen[j]:=trim(trailing ',' from array_gen[j]);
                array_gen[j]:=array_gen[j] || '}';

   END LOOP;

    -- genero el nombre de la tabla donde se almacenarán los datos
    tabla := tablename || '_desc';

     -- realizo la inserción
     EXECUTE 'INSERT INTO ' || tabla ||'
(fecha_hora,causa,valor,fvr,fvs,fvt,fiu,fiv,fiw,vfr_int2,vfs_int2,vft_int2,ifu_int2,ifv_int2,ifw_int2) values ('''
      || fecha_hora || ''',' || vector[2] || ',' || vector[3]  || ',' || vector[4]  || ',' || vector[5]  || ','
      || vector[6] || ',' || vector[7] || ',' || vector[8] || ',' || vector[9] || ',''' || array_gen[0] || ''','''' || array_gen[1] || ''','''
      || array_gen[2] || ''','''' || array_gen[3] || ''','''' || array_gen[4] || ''','''' || array_gen[5] ||''' )';

return;

END; $BODY$
  LANGUAGE 'plpgsql' VOLATILE;

ALTER FUNCTION eocen_alarmas.cyc_view2(tablename "varchar", vector _float8, fecha_hora timestamptz)
OWNER TO eocen;
```

Como se observa, lo que hace este código, que toma como parámetros el nombre de la tabla en la que se ha insertado el vector, la fecha de la inserción y el propio vector, es recorrer el array sacando los datos intercalados en 6 vectores, hasta llegar al final. Una vez finalizado el recorrido, prepara la sentencia SQL y realiza la inserción en la tabla *cyc_type_desc*

Imagen 5: Proceso de descomposición de un cyclic

Para hacer efectivo este trigger, hay que insertarlo como una regla en las tablas que heredan de *cyc_type*, que se llamarán tv_xxx_cyc, dónde "tv" es el identificador de parque y "xxx" el número del aerogenerador. Para ello usamos el siguiente código:

```
-- Table: eocen_alarmas.tv_xxx_cyc

-- DROP TABLE eocen_alarmas.tv_xxx_cyc;

CREATE TABLE eocen_alarmas.tv_xxx_cyc
(
  CONSTRAINT tv_002_cyc_pkey PRIMARY KEY (fecha_hora)
) INHERITS (eocen_alarmas.cyc_type)
WITHOUT OIDS TABLESPACE eocen_escrit;
ALTER TABLE eocen_alarmas.tv_xxx_cyc OWNER TO eocen;


-- Rule: "cyc002 ON eocen_alarmas.tv_xxx_cyc"

CREATE OR REPLACE RULE cycxxx AS
    ON INSERT TO eocen_alarmas.tv_xxx_cyc DO INSTEAD  SELECT
eocen_alarmas.cyc_view2('eocen_alarmas.tv_xxx_cyc'::character varying,
new.valor, new.fecha_hora) AS cyc_view2;
```

2.2.4. Tablas de eocen_calidad

Este esquema contiene las tablas necesarias para guardar los datos obtenidos de los estudios de calidad realizados por el módulo de calidad.

De nuevo, la estructuración del esquema será una tabla por cada aerogenerador que hereda de una tabla padre *cal_type.* Cada fila de esta tabla almacena todos los datos

referentes a un único armónico: número del mismo, valores que toman para dicho armónico Vr, Vs y Vt, y los 15 interarmónicos, los que se encuentran entre ese armónico y el siguiente, de cada una de las tensiones.

Como los estudios de calidad abarcan los 50 primeros armónicos de cada señal junto con sus interarmónicos, necesitaremos insertar 50 filas en la tabla por cada estudio que queramos almacenar. Además, para facilitar la posterior extracción de datos, se impuso que la inserción se hiciera de manera consecutiva, es decir, que siempre después de la fila correspondiente a los datos del primer armónico encontraremos la que contiene los del segundo armónico, y detrás de esta la del tercero, y así sucesivamente hasta llegar al número 50.

Como estas tablas tienen un bajo número de inserciones a la vez que un elevado número de consultas, se asignaron al tablespace *eocen_lect*.

El código SQL correspondiente a la clase padre es el siguiente:

```
-- Table: eocen_calidad.cal_type

CREATE TABLE eocen_calidad.cal_type
(
  fecha_hora timestamptz NOT NULL,        -- fecha del estudio
  num_armonico int2 NOT NULL,             -- número del armónico
  armonico_vr float4 NOT NULL,            -- valor del armónico de Vr
  intarm_01_vr float4 NOT NULL,           -- valor del interarmónico 1 de Vr
  intarm_02_vr float4 NOT NULL,           -- valor del interarmónico 2 de Vr
  intarm_03_vr float4 NOT NULL,           -- valor del interarmónico 3 de Vr
  intarm_04_vr float4 NOT NULL,           -- valor del interarmónico 4 de Vr
  intarm_05_vr float4 NOT NULL,           -- valor del interarmónico 5 de Vr
  intarm_06_vr float4 NOT NULL,           -- valor del interarmónico 6 de Vr
  intarm_07_vr float4 NOT NULL,           -- valor del interarmónico 7 de Vr
  intarm_08_vr float4 NOT NULL,           -- valor del interarmónico 8 de Vr
  intarm_09_vr float4 NOT NULL,           -- valor del interarmónico 9 de Vr
  intarm_10_vr float4 NOT NULL,           -- valor del interarmónico 10  de Vr
  intarm_11_vr float4 NOT NULL,           -- valor del interarmónico 11 de Vr
  intarm_12_vr float4 NOT NULL,           -- valor del interarmónico 12 de Vr
  intarm_13_vr float4 NOT NULL,           -- valor del interarmónico 13 de Vr
  intarm_14_vr float4 NOT NULL,           -- valor del interarmónico 14 de Vr
  intarm_15_vr float4 NOT NULL,           -- valor del interarmónico 15 de Vr
  armonico_vs float4 NOT NULL,            -- valor del armónico  de Vs
  intarm_01_vs float4 NOT NULL,           -- valor del interarmónico 1 de Vs
  intarm_02_vs float4 NOT NULL,           -- valor del interarmónico 2 de Vs
  intarm_03_vs float4 NOT NULL,           -- valor del interarmónico 3 de Vs
  intarm_04_vs float4 NOT NULL,           -- valor del interarmónico 4 de Vs
  intarm_05_vs float4 NOT NULL,           -- valor del interarmónico 5 de Vs
  intarm_06_vs float4 NOT NULL,           -- valor del interarmónico 6 de Vs
  intarm_07_vs float4 NOT NULL,           -- valor del interarmónico 7 de Vs
  intarm_08_vs float4 NOT NULL,           -- valor del interarmónico 8 de Vs
  intarm_09_vs float4 NOT NULL,           -- valor del interarmónico 9 de Vs
```

```
  intarm_10_vs float4 NOT NULL,          -- valor del interarmónico 10 de Vs
  intarm_11_vs float4 NOT NULL,          -- valor del interarmónico 11 de Vs
  intarm_12_vs float4 NOT NULL,          -- valor del interarmónico 12 de Vs
  intarm_13_vs float4 NOT NULL,          -- valor del interarmónico 13 de Vs
  intarm_14_vs float4 NOT NULL,          -- valor del interarmónico 14 de Vs
  intarm_15_vs float4 NOT NULL,          -- valor del interarmónico 15 de Vs
  armonico_vt float4 NOT NULL,           -- valor del armónico  de Vt
  intarm_01_vt float4 NOT NULL,          -- valor del interarmónico 1 de Vt
  intarm_02_vt float4 NOT NULL,          -- valor del interarmónico 2 de Vt
  intarm_03_vt float4 NOT NULL,          -- valor del interarmónico 3 de Vt
  intarm_04_vt float4 NOT NULL,          -- valor del interarmónico 4 de Vt
  intarm_05_vt float4 NOT NULL,          -- valor del interarmónico 5 de Vt
  intarm_06_vt float4 NOT NULL,          -- valor del interarmónico 6 de Vt
  intarm_07_vt float4 NOT NULL,          -- valor del interarmónico 7 de Vt
  intarm_08_vt float4 NOT NULL,          -- valor del interarmónico 8 de Vt
  intarm_09_vt float4 NOT NULL,          -- valor del interarmónico 9 de Vt
  intarm_10_vt float4 NOT NULL,          -- valor del interarmónico 10 de Vt
  intarm_11_vt float4 NOT NULL,          -- valor del interarmónico 11 de Vt
  intarm_12_vt float4 NOT NULL,          -- valor del interarmónico 12 de Vt
  intarm_13_vt float4 NOT NULL           -- valor del interarmónico 13 de Vt
  intarm_14_vt float4 NOT NULL,          -- valor del interarmónico 14 de Vt
  intarm_15_vt float4 NOT NULL           -- valor del interarmónico 15 de Vt
)
WITHOUT OIDS TABLESPACE eocen_escrit;
ALTER TABLE eocen_calidad.cal_type OWNER TO eocen;
```

Las tablas que hereden de *cal_type* se llamarán *tv_xxx_cal_10mp*, donde “tv” es el identificador del parque, “xxx” el número de la máquina y 10mp hace referencia a que los estudios son diezminutales (esto se especifica por si en un futuro se decide ampliar el tipo de los estudios a otras modalidades propuestas en el RD).

2.2.5. Tablas de eocen_parque

Este esquema se diseñó para mantener los valores de las distintas energías (activa, reactiva y consumida) del parque, que se obtienen a partir de las de todos los aerogeneradores que componen el parque.

Para ello, se definió una tabla padre *parque_type* con los campos necesarios de la que heredan las diversas tablas de cada parque, que siguen la nomenclatura tv_xxx_prq, donde “tv” es el identificador del parque. Por ser tablas destinadas a la inserción continua de datos, todas ellas se asignaron al tablespace *eocen_escrit*.

El código del script de la tabla padre es el siguiente:

```
-- Table: eocen_parque.parque_type

CREATE TABLE eocen_parque.parque_type
(
  fecha_hora timestamptz NOT NULL,       -- fecha de los datos
  eac_actu float4 NOT NULL,              -- energía active actual
  ere_actu float4 NOT NULL,              -- energía reactiva actual
  con_actu float4 NOT NULL               -- energía consumida actual
)
WITHOUT OIDS TABLESPACE eocen_escrit;
ALTER TABLE eocen_parque.parque_type OWNER TO eocen;
```

Además, para poder realizar los cálculos acumulativos sobre las energías diarias, semanales, mensuales y anuales, se definió una función PL/SQL que, dadas 2 fechas, convierte el intervalo a horas. El código de dicha función es:

```
-- Function: eocen_parque.horas(timestamptz, timestamptz)


CREATE OR REPLACE FUNCTION eocen_parque.horas(timestamptz, timestamptz)
  RETURNS float8 AS

$BODY$select date_part('days',$1-$2)*24 + date_part('hours',$1-$2);$BODY$
  LANGUAGE 'sql' VOLATILE;

ALTER FUNCTION eocen_parque.horas(timestamptz, timestamptz) OWNER TO eocen;

COMMENT ON FUNCTION eocen_parque.horas(timestamptz, timestamptz) IS
'conversion de un intervalo en horas';
```

3. Sistema de visualización de los datos: SCWEOCEN

El SCWEOCEN (Sistema de Control Web EOCEN), es una aplicación WEB que permite presentar al operador todos los datos recogidos por el sistema así como realizar las tareas de telecontrol necesarias, actuando como interfaz hombre-máquina. El desarrollo de esta aplicación se apoya en el uso de múltiples tecnologías de fuentes abiertas que complementan y amplían la funcionalidad ofrecida por HTML, entre las que podemos mencionar:

- **JSP** [5]: permite la inserción de código Java en las páginas HTML, consiguiendo que las mismas sean dinámicas. Además, hace posible el acceso a bases de datos para insertar o extraer datos.

- **JavaScript** [6]: lenguaje de programación que permite la interacción de la aplicación con el usuario a través de secuencias de código que se ejecutan directamente en el equipo cliente. Para este proyecto, debido a la exigencia del usuario de que el sistema debe funcionar bajo el navegador Internet Explorer versión 6 o superior, nos vimos obligados a no cumplir este estándar de manera estricta para poder adaptarnos a las peculiaridades sobre el mismo impuestas por dicho navegador.

- **XML**: lenguaje extensible de etiquetas diseñado para separar el contenido de un archivo de su representación, facilitando el procesado de los datos que contiene. Se utiliza para el intercambio de datos con la base de datos.

- **CSS**: son las siglas de las hojas de estilos (Cascading Style Sheet). Permiten independizar el contenido de la página JSP de los atributos de estilo que se aplican a cada elemento que la compone, así como la reutilización de los estilos definidos en diversos elementos y páginas.

Además de todas estas tecnologías, SCWEOCEN utiliza una serie de motores específicos desarrollados con Java, JavaBeans y Servlets que le dotan de funcionalidades adicionales necesarias para alcanzar los objetivos marcados en las memorias descriptiva y justificativa de este proyecto.

Para desarrollar el SCWEOCEN se emplearon 2 herramientas:

- **Entorno de desarrollo Eclipse**: permite la implementación de programas en prácticamente cualquier lenguaje de programación. En este proyecto se utilizó para desarrollar las clases Java, Beans y Servlets necesarios, así como para la inclusión del

código Java en las páginas JSP, tarea para la que se contó con la ayuda del plugin Lomboz. Además, el Eclipse se empleo para la estructuración, integración y configuración del SCWEOCEN.

- **DreamWeaver**: herramienta de edición de páginas Web. Se utilizó para realizar el diseño gráfico de las distintas pantallas que componen el SCWEOCEN.

La aplicación Web resultante se instalará sobre un contenedor JSP, el Apache Tomcat, que permitirá el acceso al SCWEOCEN desde cualquier ordenador conectada a la intranet del promotor.

En este apartado se expondrá cómo se llevó a cabo la implementación de cada uno de los motores de los que requiere la aplicación para funcionar, la estructuración del sistema de visualización, la organización de la navegabilidad del mismo y la integración de todos estos desarrollos para obtener el SCWEOCEN. Así mismo, se detalla el proceso de instalación de dicha interfaz en el servidor principal del proyecto.

3.1. Desarrollo de los motores del SCWEOCEN

Como ya se ha comentado anteriormente, los motores están compuestos por Servlets, Beans y clases Java y se utilizan para aumentar las capacidades del sistema a la hora de recuperar datos de la PostgreSQL, tratarlos, formatearlos y presentarlos al operador y de enviar órdenes a los diferentes ARP.

Los motores desarrollados para el SCWEOCEN son:

- **Motor de conexión a base de datos**: su misión es liberar al resto de módulos de todas las tareas relacionadas con el acceso a la base de datos. Presenta una interfaz muy sencilla que requiere como único parámetro una cadena con una sentencia SQL y devuelve los resultados de la consulta, si los hay, a través de mapas o iteradotes por filas o columnas.

- **Motor de gráficas**: motor diseñado para representar los datos del sistema a través de gráficas de puntos (una o varias variables, cada una con su eje) o de barras (para la representación de espectros). Permite configurar prácticamente todos los parámetros

de la gráfica a representar mediante el paso de parámetros por la URL: tamaño, variables a representar, colores de cada una, dominio y rango de los ejes, etiquetas…

- **Sistema y Repositorio**: son dos Beans que mantienen respectivamente la configuración del sistema estática (estructura y características del sistema en los parques instalados) y la dinámica en tiempo de ejecución (en el dominio de la sesión).

- **Actualización de datos**: motor que, tomando como parámetros el periodo de refresco y las variables mostradas en la pantalla, se encarga de actualizar los datos contenidos en la pantalla de forma periódica. Permite especificar el formato deseado para cada dato y el periodo de refresco.

- **Motor de envío de órdenes**: presenta una interfaz para el envío de órdenes de forma totalmente transparente al resto de módulos del SCWEOCEN. Se encarga de establecer la conexión con el plugin del núcleo encargado del procesado de las órdenes, y del formateo y envío en tramas de los parámetros necesarios para que dicho plugin sea capaz de generar la orden.

En este apartado se describirá el funcionamiento los distintos motores ya presentados, omitiéndose, por razones de extensión, el código fuente de los mismos. Además, se mencionarán algunas clases auxiliares adicionales también necesarias para el funcionamiento del SCWEOCEN.

3.1.1. Motor de conexión a la base de datos

Este motor esta constituido por tres clases Java que forman parte del paquete *scweocen.conexión*:

- **ObjBD.java:** Objeto que contendrá los resultados de la consulta en caso de que los hubiera. Posee dos constructores que permiten organizar los datos, tanto si se requieren por filas, a través de una lista de mapas columna – valor en la que cada mapa se contiene los datos de una fila, como si se necesitan ordenados por columnas, a través de un mapa columna – lista de valores.

  Para ello, toma como parámetro el *ResulSet* obtenido de la ejecución de la consulta, que es un conjunto de objetos, en el que cada objeto contiene una lista con los valores de una fila.

Los métodos públicos que proporciona esta clase para el acceso a los datos son los siguientes:

- hayDatos(): devuelve true si la consulta devolvió algún valor
- getFila(int i): devuelve un mapa con las parejas nombre_columna – valor para la fila indicada como parámetro
- getColumna(String nombreDeLaColumna): devuelve un iterador sobre la lista de los valores de la columna cuyo nombre se le pasa como parámetro.
- getNumeroDeColumnas(): devuelve el número de columnas resultantes de la consulta
- getNumeroDeFilas(): devuelve el número de filas resultantes de la consulta

El proceso de ordenación de los datos llevado a cabo queda recogido en el diagrama de la imagen 6.

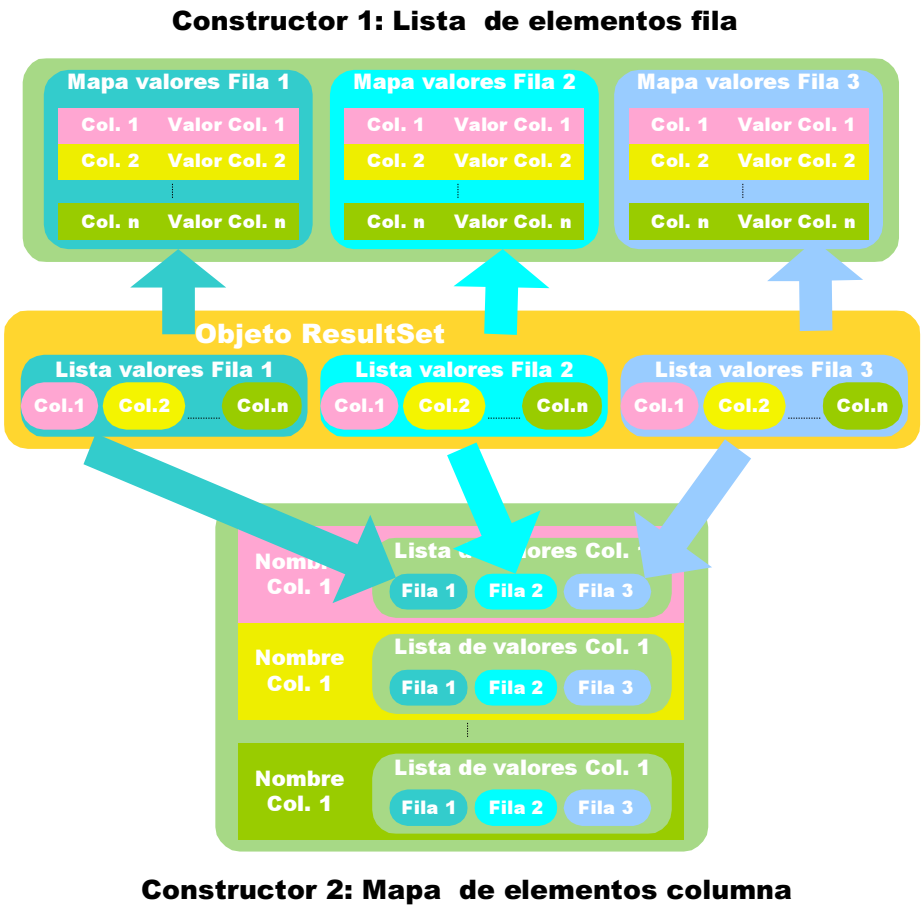


Imagen 6: Organización de los datos según el constructor utilizado

- **Conexion.java**: Superclase abstracta de los métodos encargados del establecimiento y liberación de las conexiones con la base de datos así como de la ejecución de las sentencias SQL. Define el método de conexión a la base de datos como abstracto, debiendo ser implementado por todas las clases que hereden de ella. Esto se ha hecho así para independizar el acceso de la base de datos del tipo de base de datos a la que nos conectamos: para trabajar con una base de datos distinta basta con añadir una nueva clase hija que implemente el acceso específico a dicha base de datos.

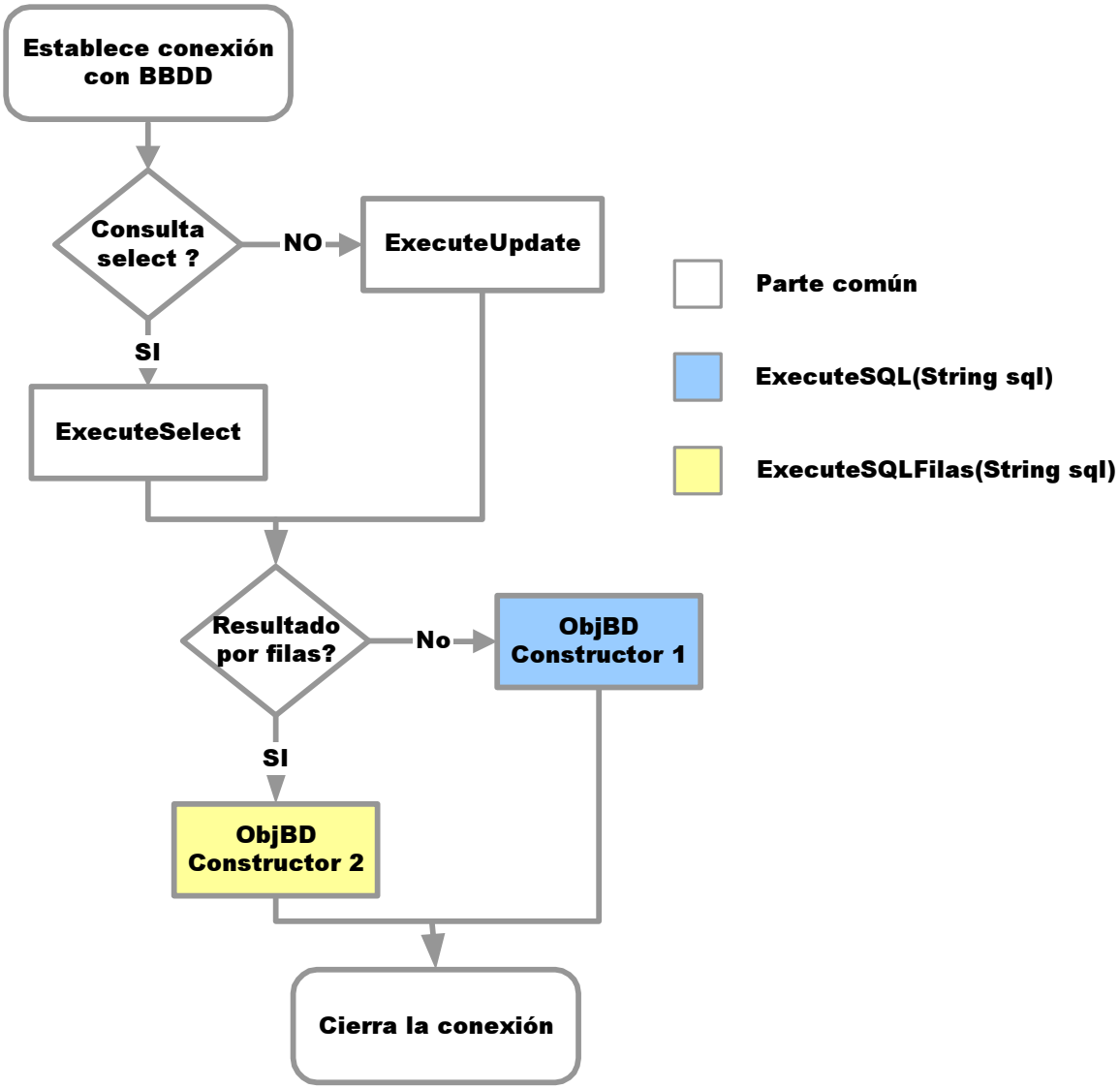


Imagen 7: Diagrama de flujo que recoge el funcionamiento de la clase Conexion.java

Esta clase posee dos métodos públicos, executeSQL(String sql) y executeSQLFilas(String sql). Ambos toman como parámetro una cadena con la sentencia SQL, ejecutan la sentencia y devuelven el resultado. La diferencia reside en

la forma de devolver los datos, ya que cada uno utiliza un constructor de *ObjBD* distinto.

En el diagrama de flujo de la imagen 7 recoge el funcionamiento de estas clases.

- **ConexiónPostgre.java**: Clase que hereda de *Conexion.java* e implementa el método de conexión específico para la base de datos PostgreSQL.

  Para permitir la conexión y acceso a la Postgre necesitamos incluir el driver de comunicaciones *jdbc.jar* específico de esta base de datos en la carpeta del Tomcat /usr/local/tomcat5/common/lib/, que contiene todas las librerías utilizables por las aplicaciones que sirve Tomcat. Además deberemos configurar los parámetros de la conexión en el archivo XML de la aplicación: driver utilizado, cadena URL para la conexión, número máximo de conexiones, y condiciones para la finalización de las mismas.

El contenido de dicho archivo es el siguiente:

```
<Context path="/SCWEOCEN" reloadable="true"
docBase="/usr/share/tomcat5/webapps/SCWEOCEN"
workDir="/usr/share/tomcat5/work">

<Resource name="jdbc/CyrDB" auth="Container" type="javax.sql.DataSource"/>

<ResourceParams name="jdbc/CyrDB">

  <parameter>
        <name>factory</name>
        <value>org.apache.commons.dbcp.BasicDataSourceFactory</value>
  </parameter>

  <!-- Maximum number of active DB connections in pool.   -->
  <parameter>
        <name>maxActive</name>
        <value>10</value>
  </parameter>

  <!-- Maximum number of idle DB connections to retain in pool.   -->
  <parameter>
        <name>maxIdle</name>
        <value>30</value>
  </parameter>

  <!-- Maximum time to wait for a DB connection to become available  in ms  -->
  <parameter>
        <name>maxWait</name>
        <value>10000</value>
  </parameter>
```

```
    <!-- DB username and password for DB connections  -->
    <parameter>
            <name>username</name>
            <value>eocen</value>
    </parameter>
    <parameter>
            <name>password</name>
            <value>eocen</value>
    </parameter>

    <!-- Class name for postgre JDBC driver -->
    <parameter>
            <name>driverClassName</name>
            <value>org.postgresql.Driver</value>
    </parameter>

    <!-- The JDBC connection url for connecting to your DB.   -->
    <parameter>
            <name>url</name>
            <value>jdbc:postgresql://192.168.104.25:5432/eocen</value>
    </parameter>

  <!-- Configuration for auto closing connections: timeout und log -->
  <parameter>
            <name>removeAbandoned</name>
            <value>true</value>
   </parameter>
   <parameter>
             <name>removeAbandonedTimeout</name>
             <value>60</value>
   </parameter>
   <parameter>
            <name>logAbandoned</name>
            <value>true</value>
   </parameter>

</ResourceParams>
</Context>
```

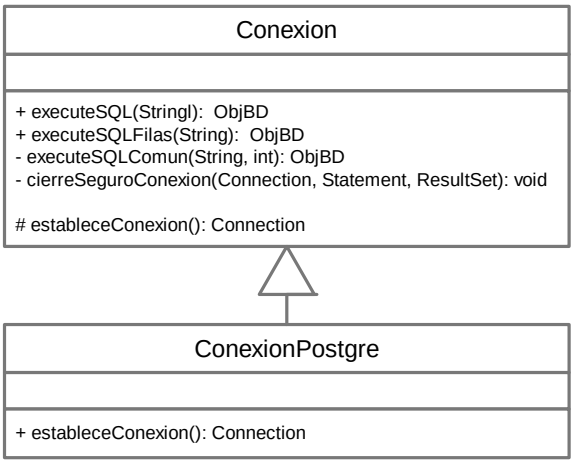


Imagen 8: Diagrama UML de las clases de conexión a la BBDD

El uso de este motor para realizar una consulta a la base de datos se reduce a crear una instancia de *ConexionPostgre*, componer la sentencia SQL a ejecutar y llamar al método *executeSQL()* o *executeSqlFilas()*, que devuelven un *ObjBD* con el resultado de la consulta. Finalmente, solo queda cerrar la conexión.

```
Conexion con = new ConexionPostgre();
ObjBD objBD = con.executeSql("Select * from…..")
con.close();
```

3.1.2. Motor de gráficas

El motor de gráficas está compuesto por el conjunto de clases y Servlets que componen el paquete *scweocen.graficas:*

- **Servlets generadores de gráficas**: *SuperGraficas.java, GraficasPuntos.java, GraficasBarras.java, GraficasPuntosEOCEN.java* y *GraficasBarrasEOCEN.java* constituyen el conjunto de Servlets encargados de generar las gráficas. El diagrama UML con las relaciones entre ellos queda recogido en la imagen 9:

  Se implementaron siguiendo una estructura de clases jerárquica, con una superclase que contiene los métodos comunes de la que heredan una clase por cada tipo de gráfica a generar (puntos y barras). Todas estas clases son totalmente genéricas y podrían ser utilizadas en cualquier proyecto, sin más que cambiar los métodos de conexión a la base de datos y tratamiento de las variables, que son específicos de la aplicación y están recogidos en las clases del último nivel de la jerarquía. El diagrama UML que recoge esta estructura queda recogido en la figura de la imagen 9.

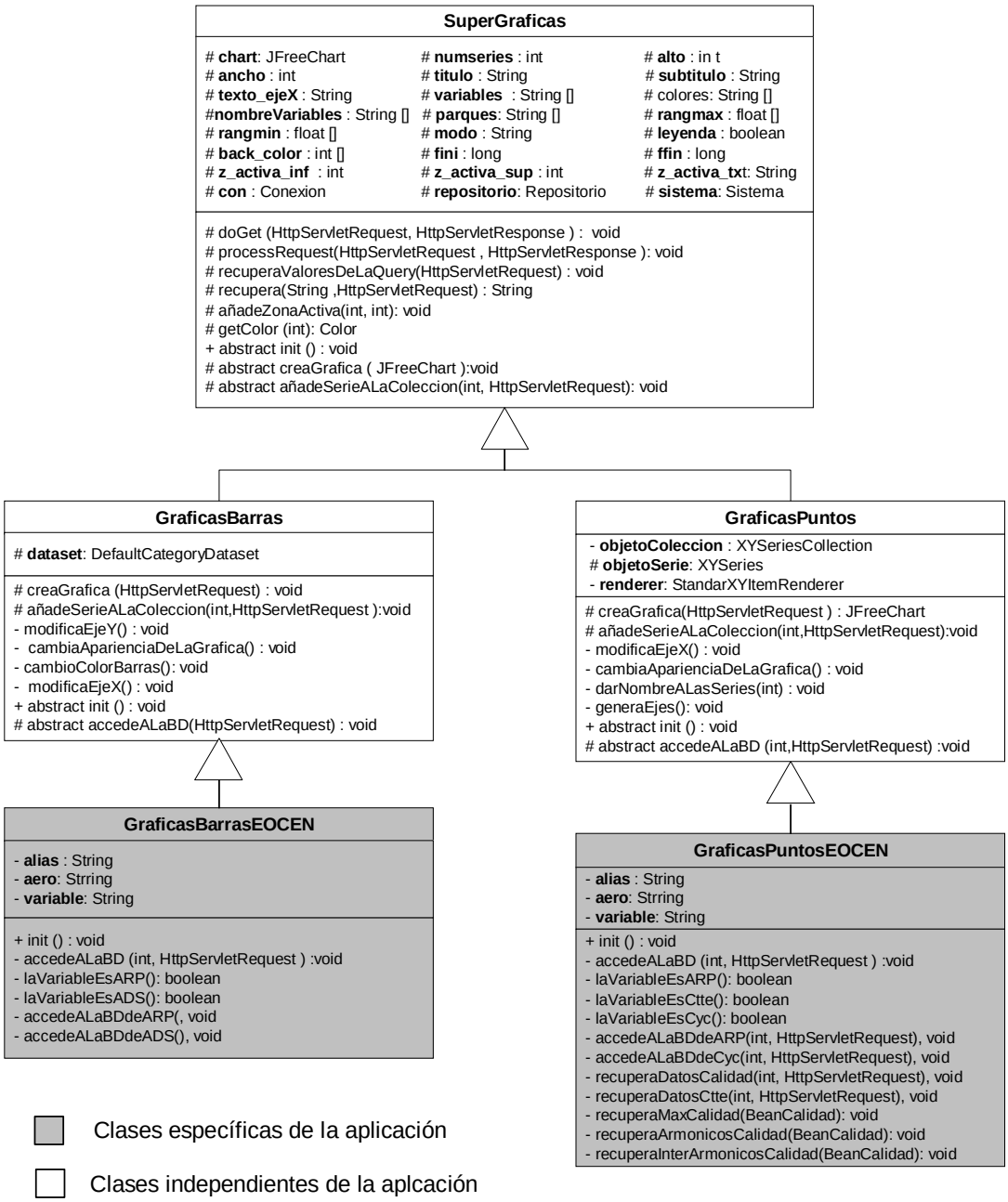


Imagen 9: Diagrama UML de los Servlets generadores de gráficas

La misión de los Servlets es liberar al resto de elementos del módulo de la generación de gráficas. Su funcionamiento queda recogido en el diagrama de flujo de la imagen 10.

Para su funcionamiento, el Servlet sólo requiere que se le pasen por la URL los parámetros necesarios para la configuración de las gráficas. Tras recuperar estos parámetros, se encarga de acceder a la base de datos para obtener las variables indicadas, introduciendo los datos de cada variable en una serie.

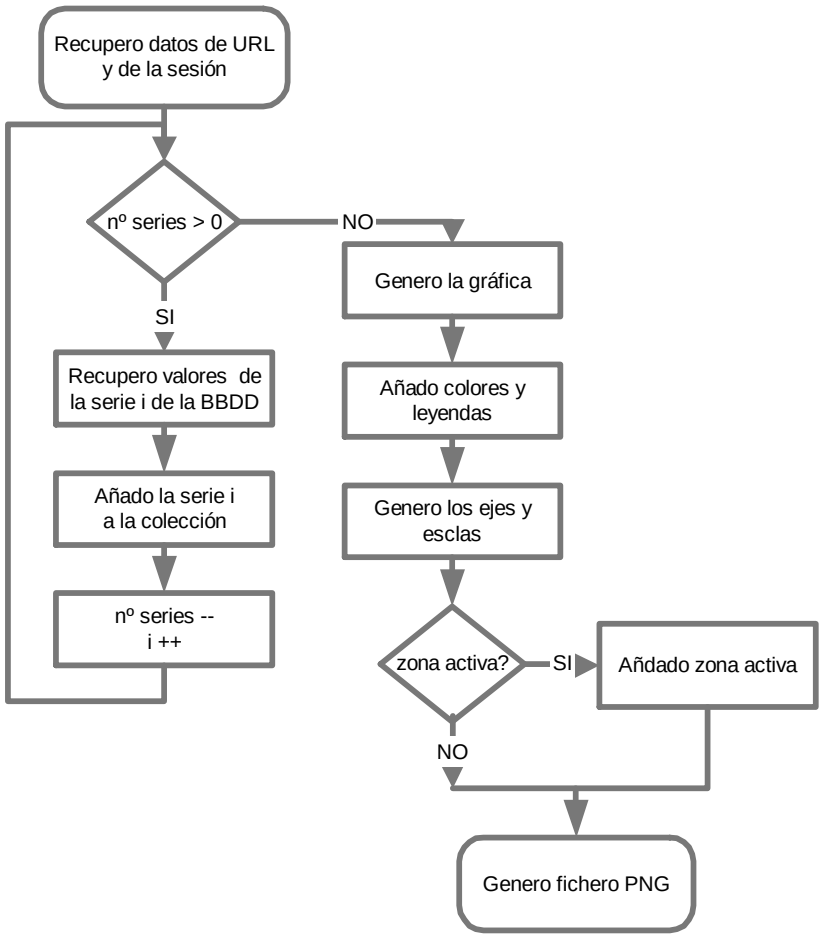


Imagen 10: Diagrama de flujo del funcionamiento de los Servlets del motor de gráficas

Una vez recuperados todos los datos, se procesan todas las series, obteniéndose la gráfica. Posteriormente, se ajustan los parámetros de color, se le añaden las leyendas, se generan los ejes y escalas para las distintas series y, si se ha solicitado, se sombrea la zona activa de la gráfica. Tras finalizar este proceso, se genera el fichero PNG con la imagen resultante.

Las funcionalidades necesarias para el funcionamiento de estos Servlets las aportan las librerías de fuentes abiertas JFreeChart v0.9.21 y JCommon v0.9.6, ambas del proyecto JFreeChart (http://www.jfree.org/jfreechart/), que presentan una interfaz sencilla pero potente para el tratamiento de datos en *datasets* y la generación de gráficas a partir de los mismos.

Para el funcionamiento de estos Servlets, además de la inclusión de las librerías anteriormente mencionadas en el proyecto, fue necesario declarar y mapear los Servlets en el archivo de configuración *web.xml* mediante la inclusión en el mismo de las siguientes líneas:

```
<servlet>
    <servlet-name>GraficasBarrasEOCEN</servlet-name>
    <servlet-class>scweocen.graficas.GraficasBarrasEOCEN</servlet-class>
</servlet>
<servlet>
    <servlet-name>GraficasPuntosEOCEN</servlet-name>
    <servlet-class>scweocen.graficas.GraficasPuntosEOCEN</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>GraficasBarrasEOCEN</servlet-name>
    <url-pattern>/GraficasBarrasEOCEN</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>GraficasPuntosEOCEN</servlet-name>
    <url-pattern>/GraficasPuntosEOCEN</url-pattern>
</servlet-mapping>
```

- **Bean VarGraficas.java:** Objeto contenedor cuya misión es almacenar las variables y parámetros de configuración del módulo de gráficas. Cada vez que se quiere generar una gráfica, se extrae dicho objeto del contexto de sesión, se actualizan los campos necesarios (por defecto mantiene los parámetros de la última gráfica generada) y se ejecuta el método *getURL(),* encargado de serializar todos los datos que hay en el Bean en una URL, que será la que se utilice para llamar al Servlet correspondiente.

  Este Bean es, además, el encargado de realizar el mapeo entre el nombre de las variables (nombre con el que los operadores identifican a las variables físicas, por ejemplo, *Forma de Onda Vr*), con su identificador (nomenclatura con la que trabaja el sistema y la base de datos, que para el ejemplo anterior sería *arp_xx_vfr_int2*).

  Al igual que los Servlets, para permitir la reutilización del módulo en otras aplicaciones, el Bean se implementó siguiendo una estructura jerárquica con una clase abstracta padre con las variables y métodos genéricos y una clase hija que contiene los métodos específicos para esta aplicación. El diagrama UML queda recogido en la figura número 11:

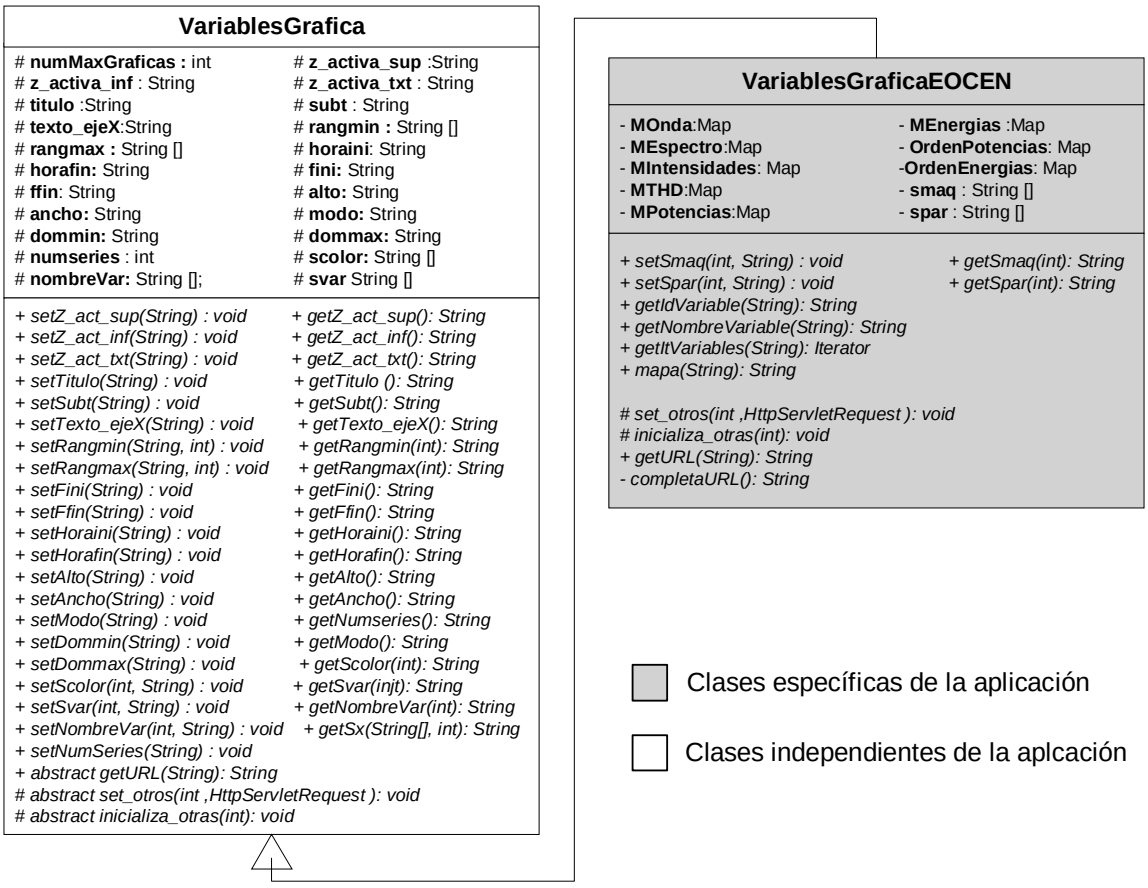


Imagen 11: Diagrama UML del Bean VarGraficas

- **ConfIniGraficas.java:** clase que contiene los parámetros de las distintas gráficas preconfiguradas junto con un único método público, *cargaConfIni(),* que toma como parámetros el Bean *VarGraficas*, el identificador de la gráfica preconfigurada a representar y algunos elementos auxiliares (Sistema y Repositorio) y se encarga de cargar los parámetros de configuración de la gráfica seleccionada en el Bean.

  La finalidad de esta clase es actuar como "fichero de configuración", permitiendo incorporar nuevas gráficas preconfiguradas al sistema sin más que añadir las líneas correspondientes en el cuerpo de la clase, sin tener que modificar el resto del módulo de gráficas.

  Para facilitar su uso, a pesar de no ser un Bean propiamente dicho, se decidió introducir esta clase como tal en el contexto de sesión de la aplicación.

El funcionamiento del módulo de gráficas queda reflejado en diagrama de la imagen 12: Cuando queremos representar una gráfica preconfigurada por pantalla, llamamos al método *cargaConfIni()* de la clase *ConfIniGraficas* con el identificador que corresponde a la gráfica deseada (paso 1), que volcará los parámetros de dicha gráfica sobre el Bean VarGraficas (paso 2). Éste generará, a partir de los datos que contiene, la URL necesaria para llamar al Servlet necesario según el tipo de gráfica (paso 3). Al insertar dicha URL en una etiqueta <img> en la página Web se realiza la llamada al Servlet (paso 4), que genera la gráfica y devuelve la imagen en lugar de la etiqueta (paso 5).

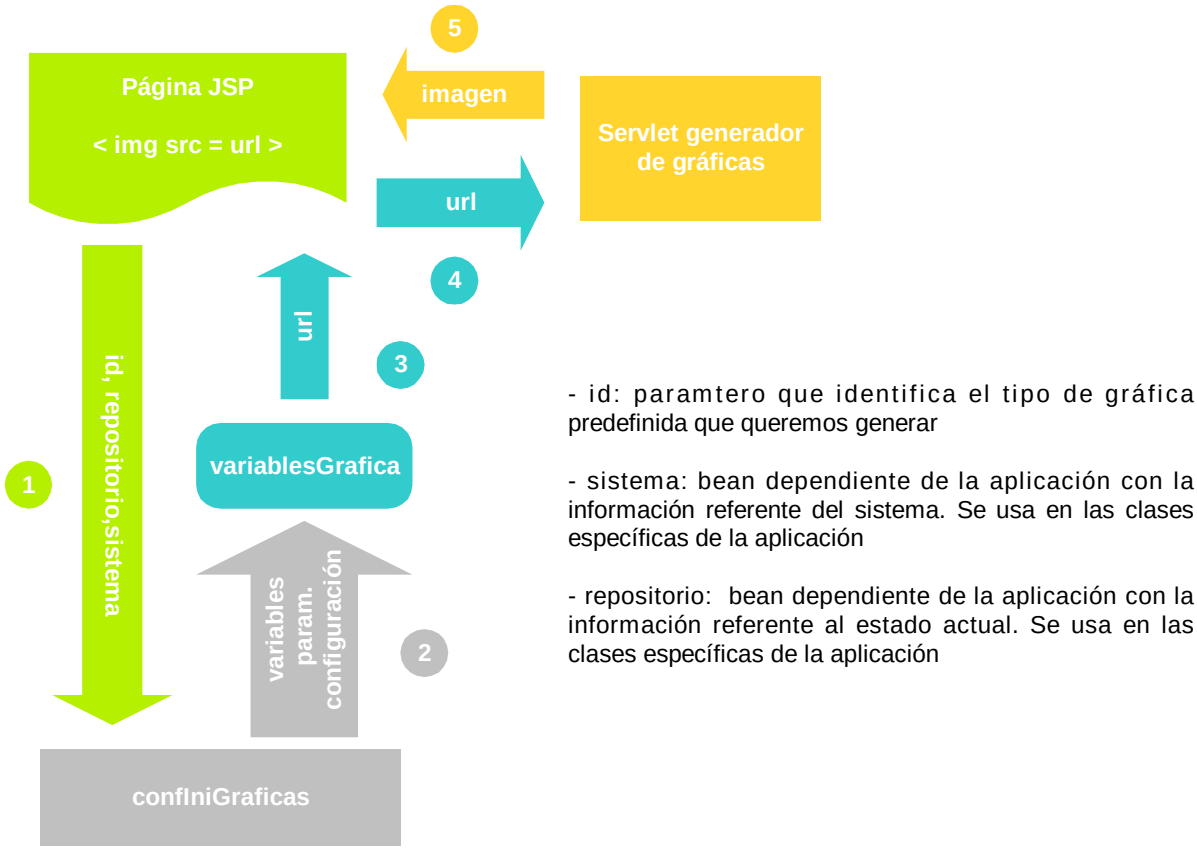


Imagen 12: Funcionamiento del módulo de gráficas

La manera elegida para implementar este proceso en una página Web es insertando un iframe del tamaño de la gráfica con la etiqueta <img>. Dicho iframe realiza por defecto la llamada a *cargaConfIni* con el identificador 0, que se corresponde con la gráfica vacía. Cuando seleccionamos una gráfica determinada, a través de funciones JavaScript se fuerza el refresco del iframe, pasándole el nuevo identificador de la gráfica, lo que pone nuevamente en marcha el proceso anteriormente descrito, dando como resultado la visualización de la gráfica

deseada. Este proceso de forzar el refresco del iframe se puede utilizar también para actualizar la gráfica mostrada por pantalla.

El módulo también permite, mediante el uso directo de la URL, representar gráficas con parámetros escogidos por el operador. Esto se consigue a través del uso de menús visuales y funciones JavaScript, que recogen los parámetros seleccionados por el operador en los menús y con ellos construyen la URL necesaria para la llamada al Servlet correspondiente, que posteriormente se inserta en la etiqueta <img>. El problema de este modo de funcionamiento es que requiere de la implementación de menús visuales ad-hoc, según los requisitos de la información a mostrar, y funciones JavaScript específicas para cada menú. Además, al no utilizarse el bean *varGraficas*, no se almacenan los parámetros de configuración de la última gráfica seleccionada.

3.1.3. Sistema y Repositorio

El Sistema y el Repositorio son 2 Beans almacenados en el contexto de sesión de la aplicación, que contienen respectivamente la información sobre la estructuración del sistema instalado (regiones, parques que contiene cada región, subparques en que se divide cada parque, y máquinas instaladas en cada parque) y sobre la zona/máquina cuyos datos estamos monitorizando.

Estas dos clases, junto con los objetos auxiliares *Elemento* y *CjtoAtrib*, diseñados para organizar la información que contienen, forman parte del paquete *scweocen.beans.sistema.*

- **Repositorio.java**: es un objeto muy sencillo que almacena la información sobre los parámetros de la página Web actual. Los atributos que posee son:
    - maquinaActual: ARP o ADS cuyos datos se están mostrando en pantalla
    - subParqueActual: subparque en el que se encuentra ubicada la máquina
    - parqueActual: parque al que pertenece el ARP o ADS
    - nodoActual: nodo al que pertenece el parque
    - regionActual: región que contiene al parque

Para poder consultar y modificar todos estos datos, se dotó al Bean de los métodos *set* y *get* correspondientes a cada atributo.

- **CjtoAtrib.java:** Objeto que se diseñó para albergar los datos del Sistema. Está formado por un único atributo, un *TreeMap* sobre el que se irá organizando la información a almacenar. Posee varios métodos para realizar inserciones y extracciones en el mapa de diversos tipos de datos (parejas nombre del atributo – valor, en el que el valor puede ser una cadena, un entero o un lista):
  - *añadirAtributoMap(String clave, String valor):* permite almacenar un atributo de tipo cadena bajo la clave especificada.
  - *añadirAtributoEnteroMap(String clave, int valor):* permite almacenar un atributo de tipo entero bajo la clave especificada.
  - *añadirLista(String clave, List valor):* permite almacenar un atributo de tipo lista bajo la clave especificada.
  - *getAtributoMap (String clave):* devuelve el atributo (tipo cadena) identificado con la clave especificada
  - *getAtributoEnteroMap(String clave):* devuelve el atributo (tipo entero) identificado con la clave especificada
  - *getListaAtributoMap(String clave):* devuelve el atributo (tipo lista) identificado con la clave especificada
  - *getAtributo()*: devuelve el mapa completo.

- **Elemento.java:** Objeto constituido por un *CjtoAtrib* más un entero que sirve de identificador del tipo de elemento (parque, ARP, ADS…) del cual albergamos la información. Los identificadores de los distintos tipos de Elementos se definen en el archivo *Constantes.java* (*scweocen.comun*):

```
public static final int REGION = 1;
public static final int NODO = 2;
public static final int PARQUE = 3;
public static final int SUBPARQUE = 4;
public static final int ADS = 5;
```

```
public static final int ARP = 6;
```

Posee 6 métodos públicos para añadir o consultar atributos, que se mapean en los métodos de *CjtoAtrib* y uno nuevo, *getTipoObjeto()* para consultar el tipo de elemento.

- **Sistema**: Bean encargado de almacenar toda la información estática del sistema EOCEN. Los datos de este Bean se cargan de las tablas de *eocen_conf* de la base de datos al iniciarse la sesión de un usuario.

  La información se estructura en un atributo del tipo List, que contiene una lista de *Elementos* del tipo Región. Cada uno de estos elementos esta compuesto por un nombre, un identificador, información referente a la región y una lista de *Elementos* tipo Nodo. A su vez, cada uno de los objetos Nodo posee varios atributos con el nombre e información del nodo y una lista de *Elementos* tipo Parque. Esta estructura recursiva de almacenaje se va repitiendo para los distintos niveles de jerarquía que componen el sistema: región, nodo, parque, subparque, ARP y ADS, tal y como se puede observar en la figura 13.

  Para poder acceder a cualquiera de los *Elementos* y extraer información de él, se definió un método público, *getElemento(int,String),* que toma como parámetros el tipo de *Elemento* a recuperar y los identificadores de los niveles de jerarquía superior y se encarga de localizar el *Elemento* indicado dentro de la estructura.

El acceso a cualquiera de los 2 Beans se puede realizar desde cualquier página Web de la aplicación sin más que recuperarlos del contexto de sesión de la misma, operación que se realiza mediante la inclusión de las siguientes líneas en el código en la página:

```
<jsp:useBean id="repositorio" class="scweocen.beans.Repositorio" scope="session" />
<jsp:useBean id="sistema" class="scweocen.beans.Sistema" scope="session" />
```

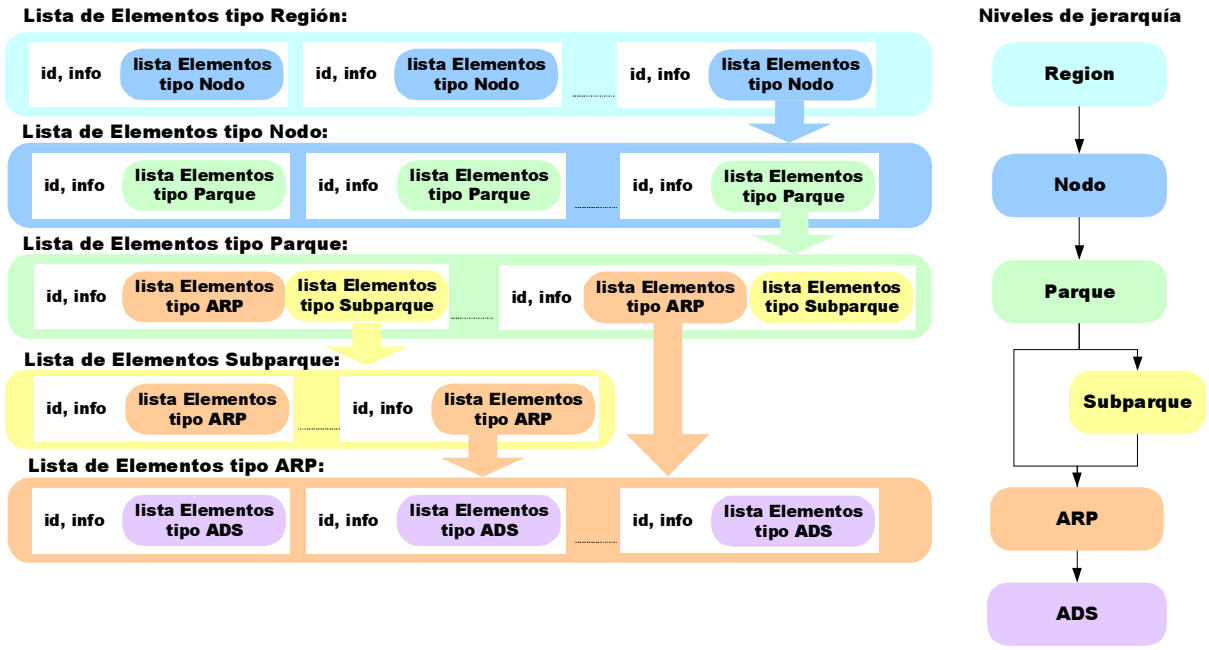


Imagen 13: Organización de la información del Sistema

3.1.4. Motor de actualización de datos

La misión de este motor consiste en refrescar periódicamente los datos mostrados por pantalla sin necesidad de tener que forzar el refresco de la pantalla. Esto se consigue abriendo una conexión en background mediante JavaScript. A través de esta conexión se realiza la consulta a la base de datos y se obtienen las variables requeridas en un archivo XML, que posteriormente se muestran en pantalla gracias a las funcionalidades aportadas por JavaScript.

Para poder realizar el proceso anteriormente descrito, fue necesario combinar las prestaciones de las tecnologías JSP, Servlet, XML y JavaScript. Los elementos que componen este motor son:

- **Servlet CargarDatos:** Servlet cuya misión es recuperar los datos de la base de datos. Debido a la diversidad de variables a recoger para cada página, almacenadas en diferentes esquemas, y por lo tanto de consultas SQL diferentes a generar, resulta muy complicado implementar un Servlet genérico, por lo que se decidió desarrollar un

Servlet específico para cada página. Todos ellos están contenidos en el paquete *scweocen.datos:*

- CargarDatosPrincipal.java : para los datos de la página principal
- CargarDatosEnergías.java : para la página de contadores de energía
- CargarDatosElemento.java: para las páginas de elementos

Cada vez que el Servlet recibe una solicitud, construye las sentencias SQL necesarias para obtener los datos de la página, la ejecuta y con los resultados genera un mapa de parejas *nombre_variable – último_valor,* que se introduce en el objeto *Request* de la petición. Finalmente, redirige la petición a la página *generadorXML.jsp*

Para que los Servlets puedan funcionar, deben declararse y mapearse en el archivo web.xml:

```
<servlet>
    <servlet-name>CargarDatosPrincipal</servlet-name>
    <servlet-class>scweocen.datos.CargarDatosPrincipal</servlet-class>
</servlet>
<servlet>
    <servlet-name>CargarDatosElemento</servlet-name>
    <servlet-class>scweocen.datos.CargarDatosElemento</servlet-class>
</servlet>
<servlet>
    <servlet-name>CargarDatosEnergias</servlet-name>
    <servlet-class>scweocen.datos.CargarDatosEnergias</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>CargarDatosPrincipal</servlet-name>
    <url-pattern>/CargarDatosPrincipal</url-pattern>
  </servlet-mapping>
 <servlet-mapping>
    <servlet-name>CargarDatosElemento</servlet-name>
    <url-pattern>/CargarDatosElemento</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
    <servlet-name>CargarDatosEnergias</servlet-name>
    <url-pattern>/CargarDatosEnergias</url-pattern>
 </servlet-mapping>
```

- **Página *generadorXML.jsp:*** página que recibe como parámetro a través del objeto *Request* un mapa formado por parejas *id_variable – último_valor* y genera a partir de ellas un fichero XML con el siguiente formato:

```
<pantalla>

        <elemento id="id_variable1">valo_variable1</elemento>
        <elemento id="id_variable2">valor_variable2</elemento>
        …

</pantalla>
```

- **Archivo xmlhttpGet.js:** archivo de código JavaScript que contiene las funciones necesarias para realizar la llamada al Servlet correspondiente y posteriormente procesar y actualizar en la página los datos recibidos en el XML:

    - *creaArray(vector):* función que toma como parámetro un vector que contienen los identificadores de las variables a actualizar y realiza una copia de las mismas en una variable local del archivo JavaScript.

    - *pedirDatos(nombre_servlet):* función que toma como parámetro el Servlet al que debe realizar la petición. Se encarga de realizar la petición al Servlet y procesar los datos recibidos como respuesta. Para ello se crea un objeto ActiveX y se le asocia un manejador que se activa al recibir la respuesta del servidor y que es el encargado de buscar las etiquetas en la página Web y sustituir los valores antiguos por los nuevos. Una vez asociado el manejador, solo le queda realizar la llamada al Servlet.

A partir de estos elementos, el proceso para permitir actualizar los datos es el siguiente:

En cada página JSP se muestran los datos que queremos que se actualicen en campos con una etiqueta *id* igual al nombre corto de la variable, cuyo formato es *pp_nnn_variable*, donde “pp” es el campo identificador de parque y “nnn” el número de la máquina. Por ejemplo:

```
<div id="tv_002_vfr_vrms" … >----</div>
```

Una vez determinados e identificados todos los datos a mostrar en la página, se crea en una variable vector en JavaScript que contenga todos los identificadores.

```
<script language="javascript">

        var refresco = new Array();
        refresco[refresco.length]='tv_002_vfr_vrms';
        refresco[refresco.length]='tv_002_vfs_vrms';
        refresco[refresco.length]="tv_002_vft_vrms';
        refresco[refresco.length]="tv_002_vrs_vrms';
        ....

</script>
```

El último paso es introducir, en la etiqueta *<body>,* que se procesa al cargar la página, la llamada a las funciones JavaScript *creaArray(vector)* y *pedirDatos(),* y programar una llamada a la función *pedirDatos()* cada 5 segundos. Para que estos métodos estén disponibles, debemos incluir en la página el fichero xmlhttpGet.js.

```
<script language="javascript" src="../include/xmlhttpGet.js"></script>
...
<body  onLoad="creaArray(refresco);pedirDatos(); window.setInterval('pedirDatos()',5000);" scroll=no>
```

El método *creaArray(vector)* se encarga de cargar el vector con los identificadores de las variables en la clase *xmlhttpGet.js.* A partir de entonces, la clase está lista para empezar a funcionar.

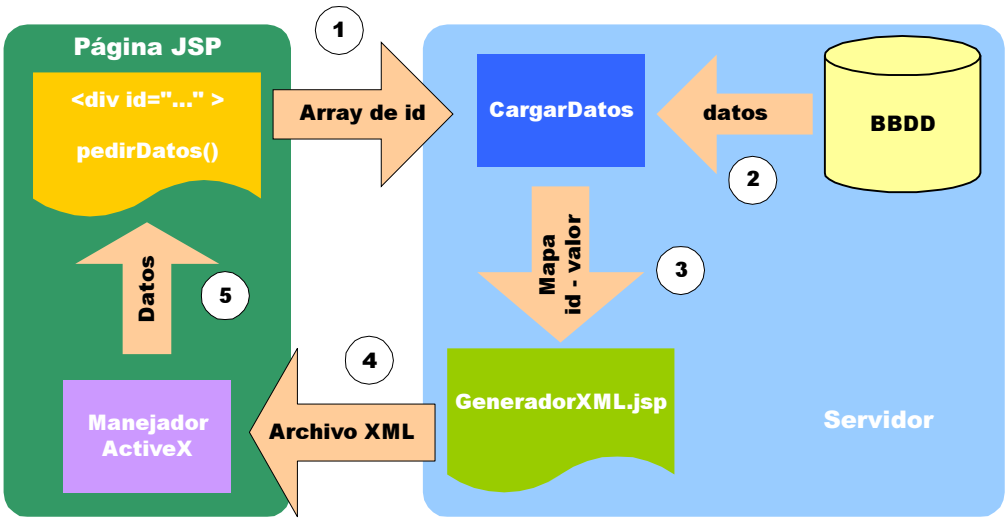


Imagen 14: Procedimiento de actualización de los datos de una página

En el momento de cargar la página, y cada 5 segundos, se produce la llamada a *pedirDatos(),* que inicia el proceso de recuperación, realizando la llamada en background al Servlet *cargarDatos* correspondiente (paso1). El Servlet accede a la base de datos para recuperar los valores solicitados (paso 2) y redirige la petición a *generadorXML.jsp* (paso 3), que con los resultados obtenidos genera un archivo XML (paso 4).

El archivo XML resultante es parseado mediante las funciones JavaScript del fichero *xmlhttpGet.js*, localizándose las etiquetas de la página que contienen cada variable a través de su *id* y cambiando el parámetro *value* de las mismas por el valor extraído del archivo XML (paso 5).

3.1.5. Motor de envío de órdenes

EOCEN se diseño como un sistema de control integral. Esto no sólo implica la monitorización de los parámetros recogidos por los ARP y ADS instalados en cada aerogenerador, sino que además debía permitir al operador el envío de determinadas órdenes a cada una de las máquinas para regular su funcionamiento.

La generación de las tramas de órdenes a enviar a las máquinas, así como su contenido quedan recogidos en el protocolo de comunicaciones y son misión del driver de comunicaciones.

Para permitir al operador enviar determinadas órdenes a los ARP y ADS, se diseñó e implemento el motor de envío de órdenes, cuya misión es, desde una página Web del SCWEOCEN, recoger los parámetros requeridos por el driver de comunicaciones para poder formar la orden, abrir una conexión en background con el plugin del núcleo encargado de recibir las órdenes y transmitirle los parámetros de la orden.

El motor consta de una clase JavaScript y de un Servlet:

- ***envioOrdenes.js:*** clase JavaScript que se encarga de generar la URL y hacer la llamada al Servlet en background.

  Posee un método *enviaOrden(orden, máquina, puertas, valores)* que toma como parámetros el identificador de la orden, utilizado por el driver de comunicaciones para discriminar la orden a enviar y saber cuántos parámetros lleva la petición, el número de máquina a la que va dirigida la orden y dos arrays que contendrán respectivamente las

puertas (nombres de los parámetros de la orden) y sus valores. Con todos estos elementos, se encarga de construir una URL cuyo formato es:

../EnvioOrden?param=campo1@campo2@campo3

- EnvioOrden: Servlet encargado del envío de la orden
- Campo1: contiene el identificador de la orden y el número de la máquina separados por un carácter almohadilla
- Campo2: lista con los nombres de las puertas separadas por el carácter almohadilla
- Campo3: lista de los valores de las puertas (en el mismo orden en que se introdujeron en el campo 2) separados por el carácter almohadilla

../EnvioOrden?param=id_orden#maq@nombre_param1#nombre_param2 …
@valor_param1#valor_param2 …

Una vez formada la URL necesaria para realizar la llamada al Servlet y tras pedir la confirmación al operador para mandar la orden, crea un objeto ActiveX y realiza la petición a través de él.

- **Servlet enviaOrdenes** (paquete *scweocen.ordenes*): Este Servlet es el encargado de trasladar las órdenes introducidas por el operador al módulo de órdenes del núcleo. Para ello, cada vez que recibe una petición comienza verificando si está completa, es decir, si contiene todos los parámetros necesarios para el tipo de orden que se especifica. Una vez hecho esto crea una URL para cada parámetro a transmitir y otra para el identificador de la orden con el siguiente formato:

http://192.168.104.25:2005/set.html?nombre=XXX&valor=YYYY

  - Nombre: nombre de la puerta. Estos son:
    - Para la URL del identificador de orden el nombre tiene el siguiente formato: E_AN_TR_TV_NNN_PET_XX_XXX_TIPO, donde “E” es el

identificador del sistema (España), “AN” el de la región (Andalucía), “TR” el del nodo (Tahivilla), “TV” el del parque (Tahivilla) y “NNN” es el número de la máquina a la que va dirigida la orden.

- Para la URL de los parámetros el nombre tiene el siguiente formato: E_AN_TR_TV_MMM_PET_XX_XXX_PAR1, donde el último campo indica el número del parámetro. Así, el del segundo parámetro terminaría en PAR2, el del tercero en PAR3 y sucesivamente.

o Valor: valor que toma la puerta indicada en el nombre

Una vez construidas todas las URL, el Servlet va creando y abriendo una conexión HTTP por cada una de ellas y abriéndola, con lo que los valores se van transmitiendo al modulo de órdenes. El orden de las conexiones es el preestablecido por dicho módulo: en primer lugar, caso de haberlos, los parámetros de la orden y finalmente, el identificador de la orden.

Para que el Servlet pueda funcionar, se declaró y mapeo en el archivo *web.xml*:

```
<servlet>
        <servlet-name>EnvioOrden</servlet-name>
        <servlet-class>scweocen.ordenes.EnvioOrden</servlet-class>
</servlet>

<servlet-mapping>
        <servlet-name>EnvioOrden</servlet-name>
        <url-pattern>/EnvioOrden</url-pattern>
</servlet-mapping>
```

Haciendo uso de estos métodos, la implementación del envío de órdenes se reduce a, mediante funciones JavaScript, recoger los parámetros de la orden, crear el array de puertas y valores, y realizar la llamada al método JavaScript *enviaOrden* El proceso queda recogido en la imagen número 15.

Los identificadores de los tipos de orden definidos por el driver de comunicaciones son:

Identificador	Orden
1	Petición de puesta a cero de los contadores de
2	Petición de cyclic al ARP
3	Petición de estudio de calidad de 3 seg.

4	Petición de parada de transmisión del estudio de
5	Petición de sincronización del ARP
6	Petición de configuración de eventos
7	Configuración del origen de datos
8	Configuración de las relaciones de transformación
10	Configuración de eventos de almacenamiento en

Tabla 2: Identificadores de las órdenes que puede enviar el driver de comunicaciones

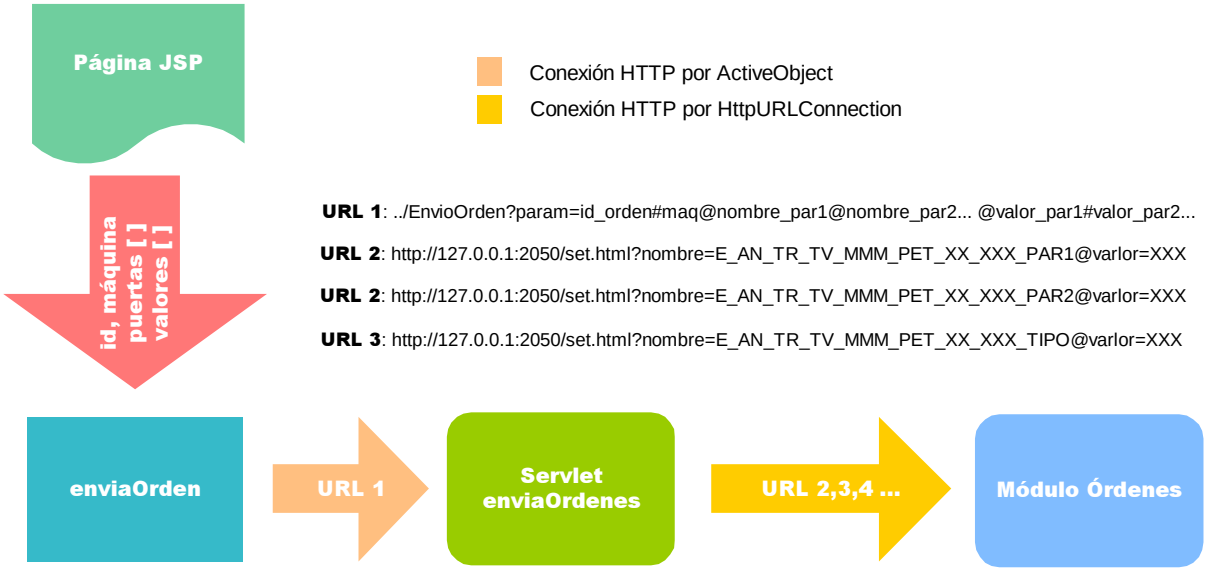


Imagen 15: Esquema de funcionamiento del módulo de envío de órdenes

3.1.6. Otras clases auxiliares

Durante el diseño del SCWEOCEN, se observó que la mayor parte de las páginas que componen la aplicación requerían de una serie de funcionalidades comunes. Para evitar tener que implementarlas varias veces en el código local de cada página, se decidió crear un paquete, el *scweocen.comun*, que contuviera las clases que aportan estas funcionalidades y desde el que pudiesen ser importadas para su uso en cualquier página.

Las clases que componen este paquete son:

- ***Depurador.java***: clase que se encarga de organizar la salida por consola de los distintos mensajes de error que se producen durante el funcionamiento de la aplicación.

Define seis niveles de depuración (errores de graficas, de órdenes, de acceso a base de datos, en la ejecución de los Servlets *CargarDatos* y en el Bean de calidad y, salidas de los bloques *try catch),* que pueden ser activados o desactivados mediante la modificación de las constantes booleanas de la clase.

Posee un método público sobrecargado que toma como parámetro el tipo de nivel de depuración y el mensaje de error o en su lugar un objeto del tipo *Exception*. Al ser llamado este método comprueba si el tipo de depuración seleccionado está activado y, en caso afirmativo, imprime el mensaje por consola.

- ***Formateo.java***: clase que contiene los métodos necesarios para el formateo de los datos así como para realizar los castings de un tipo de dato a otro:
  - Métodos para trabajar con fechas que permiten:
    - A través del uso de un objeto tipo *SimpleDateFormat,* esta clase permite obtener de los objetos tipo *Date* una cadena con el día, mes, año, hora, minutos, segundos o milisegundos.
    - La adición de un periodo de tiempo a la fecha representada por el objeto *Date* devolviendo el resultado en un objeto tipo *Second*.
    - La conversión de las cadenas que representan la fecha en formato dd/mm/yyyy al formato requerido por la base de datos: mm/dd/yyyy
    - La conversión de cadenas que contiene fechas a su equivalente en milisegundos (dato tipo long)
  - Métodos para trabajar con decimales que permiten:
    - Ajustar cualquier número a una cantidad determina de decimales.
    - Separar la parte entera de la decimal
    - Convertir un número almacenado en un int, long o double a string mediante el uso de un objeto tipo *DecimalFormat*.
- ***Constantes.java*:** Contenedor diseñado para albergar todas las constantes necesarias para el funcionamiento de las clases Java y páginas JSP. No posee atributos ni

métodos, ya que la información almacenada es estática y no se requiere de ningún tipo de procesado sobre la misma.

3.2. Implementación del sistema de visualización

Una vez desarrollados todos los motores necesarios para dotar al SCWEOCEN de la funcionalidad necesaria para alcanzar los objetivos marcados en las memorias descriptiva y justificativa para este módulo, se procedió a la implementación de la interfaz visual.

Como ya se ha comentado anteriormente, esta interfaz es una aplicación Web que combina las tecnologías JSP, XML, JavaScript y CSS. Esta mezcla de tecnologías obliga a realizar una adecuada estructuración de todos los archivos que componen el proyecto. Para ello, utilizando el Eclipse, se generó un proyecto Tomcat y se añadieron las carpetas necesarias:

- ***documentación***: contiene los archivos de texto con la documentación de las páginas, pruebas realizadas, resultado de las mismas, problemas a corregir, evolución de la implementación…
- ***imágenes***: carpeta destinada a albergar las imágenes de la aplicación.
- ***include***: carpeta que contiene los archivos con el código de las funciones JavaScript necesarias. Para mejorar la estructuración del proyecto, se decidió crear un archivo JavaScript por cada página, con el mismo nombre que el archivo JSP correspondiente pero con la extensión “js”, con todas las funciones específicas de dicha página. Las funciones comunes a varias páginas se almacenan en archivos independientes.
- ***pages***: Carpeta con los archivos JSP que componen el SCWEOCEN
- ***styles***: Carpeta que contiene las hojas de estilo CSS utilizadas para el diseño gráfico de la aplicación.

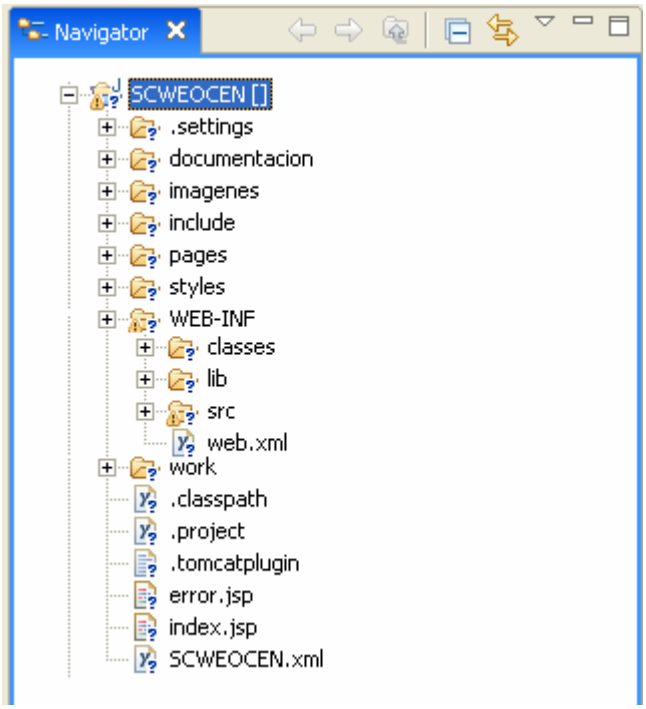


Imagen 16: Estructura organizativa del SCWEOCEN

- ***WEB-INF***: Carpeta que contiene todos los paquetes Java empleados: el código fuente de todas las clases (subcarpeta *src*), los archivos de extensión *.class* de las distintas clases ya compiladas (subcarpeta *classes*) y las librerías utilizadas (subcarpeta *lib*). Además contiene el archivo *web.xml* en el que se declaran y mapean los Servlets utilizados, se indican los drivers utilizados para las conexiones a las bases de datos y se indican cuales son las páginas de inicio y error de la aplicación:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>

 <!— ========== DECLARACIÖN DE LOS SERVLETS ================ -->

   <servlet>
      <servlet-name>CargarDatosPrincipal</servlet-name>
      <servlet-class>scweocen.datos.CargarDatosPrincipal</servlet-class>
   </servlet>
   …..

<!— ========== MAPEO DE LOS SERVLETS ====================== -->

<servlet-mapping>
      <servlet-name>CargarDatosPrincipal</servlet-name>
      <url-pattern>/CargarDatosPrincipal</url-pattern>
   </servlet-mapping>
 ….

<!— ========== PÁGINA DE INICIO ============================== -->
```

```
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

<!— ========== PÁGINA DE ERROR ================================ -->

    <error-page>
        <error-code>404</error-code>
        <location>/error.jsp</location>
    </error-page>

    <error-page>
        <exception-type>java.lang.Exception</exception-type>
        <location>/error.jsp</location>
    </error-page>

<!— ========== DRIVER CONEXIÓN A BBDD ======================== -->

    <resource-ref>
        <description>DB Connection</description>
        <res-ref-name>jdbc/CyrDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>

</web-app>
```

- ***work***: carpeta que contiene los archivos temporales resultantes de la compilación de las clases JSP en tiempo de ejecución. Es manejada directamente por el Tomcat.

Podemos observar que, además de la estructura de carpetas comentadas, en este primer nivel de jerarquía aparecen algunos archivos de configuración manejados por el propio Eclipse así como las páginas de inicio y error de la aplicación y el SCWEOCEN.xml:

- ***.classpath***: contiene las rutas de los archivos fuente de las clases Java de la aplicación y de las librerías utilizadas:

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
    <classpathentry kind="src" path="WEB-INF/src"/>
    <classpathentry kind="lib" path="WEB-INF/lib/gnujaxp.jar"/>
    <classpathentry kind="lib" path="WEB-INF/lib/junit.jar"/>
    <classpathentry kind="lib" path="WEB-INF/lib/servlet.jar"/>
    <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
    <classpathentry kind="lib" path="WEB-INF/lib/jcommon-0.9.6.jar"/>
    <classpathentry kind="lib" path="WEB-INF/lib/jfreechart-0.9.21.jar"/>
    <classpathentry kind="output" path="WEB-INF/classes"/>
```

```
</classpath>
```

- ***.tomcatplugin* y *.proyect***: archivos internos del Eclipse que contienen la descripción del proyecto y sus propiedades

- ***index.jsp***: página que carga el Tomcat cada vez que se inicia una nueva sesión de la aplicación. Esta página llama al constructor del Bean *Sistema* y redirecciona la petición a la página *principal.jsp*:

```
<%@page language="java" contentType="text/html"%>
<jsp:useBean id="sistema" class="scweocen.beans.Sistema" scope="session" />
<% response.sendRedirect("pages/principal.jsp?parqueActual=tv"); %>
```

- ***error.jsp***: página que gestiona las excepciones JSP producidas en tiempo de ejecución, imprimiendo por pantalla la URL que ha provocado el error así como la traza del mismo.

- ***SCWEOCEN*.xml**: archivo que contiene la configuración del contexto de aplicación: los path de la aplicación y los parámetros para las conexiones a las bases de datos. El contenido de este archivo ya se explicó en el punto 3.1.1 de la presente memoria.

Una vez establecida la estructura adecuada para la organización del proyecto, se añadieron a ella todas las clases, páginas, archivos JavaScript y librerías que componen los diversos motores de la aplicación.

Con todos los preparativos ya finalizados, se pasó a la fase de desarrollo del entorno visual. En los sucesivos apartados de esta memoria se analiza cómo se implementaron las distintas páginas que componen la aplicación haciendo uso de los motores ya descritos.

3.2.1. Cabecera

Al realizar el diseño de la navegabilidad de las páginas JSP de la aplicación, se observó la necesidad de tener presente en todas ellas un único menú que permitiese de forma fácil y rápida acceder desde cualquier punto de la aplicación a las demás páginas.

Así mismo, se planteó la necesidad de buscar alguna manera de evitar tener que realizar una serie de inclusiones de archivos necesarios en todas las páginas: las hojas de estilos CSS y los archivos JavaScript con las funciones necesarias para el funcionamiento del menú (*menuBar.js*) y del motor de refresco (*xmlhttpGet.js*).

Para solucionar estos 2 problemas se decidió crear una página cabecera que contuviese todos estos elementos y que posteriormente sería insertada en el resto de páginas mediante una directiva *include*:

```
<% request.setAttribute("pagina","Principal");%>

<jsp:include page="cabecera.jsp"></jsp:include>
```

El menú de navegación de la página fue implementado a través de capas y capturadores de eventos JavaScript, que se encargan de cambiar el color de los botones al pasar el puntero del ratón por encima y de, en caso de existir, hacer visible la capa que contiene el submenú asociado. Las funciones JavaScript de este menú se encuentran en el archivo *menuBar.js*

Imagen 17: Detalle del menú desplegable de la cabecera

La cabecera presenta además, un reloj y calendario con la fecha actual, obtenidos a partir de un objeto tipo *Date* y formateados y refrescados mediante la función JavaScript *mueveReloj(),* contenida en el archivo *reloj.js*; el parque cuya información estamos

monitorizando, que se obtiene del Repositorio; y el título de la página mostrada, que le debemos pasar como parámetro al realizar la inclusión a través del objeto *Request.*

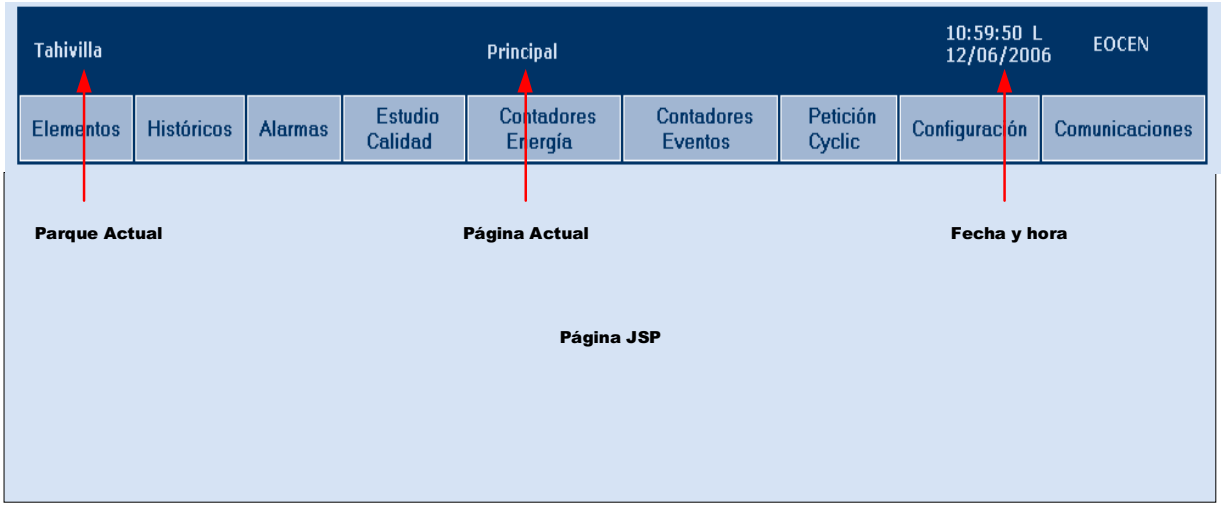


Imagen 18: Cabecera insertada en todas las páginas de la aplicación

3.2.2. Principal

Esta pantalla es la encargada de mostrar la información general del parque seleccionado. Como ya se comentó anteriormente, cada vez que se inicia una nueva sesión, la página *index.jsp* redirige la aplicación directamente hacia esta página, pasándole por la URL el parque preseleccionado cuyos datos se mostrarán (en el caso del prototipo de este proyecto, sólo hay un parque, el de Tahivilla, por lo éste será el predeterminado. Cuando se implante la aplicación en más parques, la página *index* contendrá un menú que permitirá elegir el parque deseado).

La pantalla se divide en dos partes. En primer lugar tenemos un mapa con la ubicación geográfica del parque, acompañado de un cuadro con los datos del parque. Todos estos datos son estáticos, por lo que se introdujeron a mano en el mismo código fuente de la página.

En segundo lugar, en la parte inferior izquierda, se presenta un cuadro con los datos generales del parque que muestra los resúmenes anuales, mensuales y diarios de las energías generadas por el parque eólico así como las potencias activas y reactivas consumidas. Todos los datos de este cuadro se refrescan cada 5 segundos haciendo uso del motor de refresco de datos y del Servlet *CargarDatosPrincipal* , que los recoge de las tablas de *eocen_parque* de la base de datos.

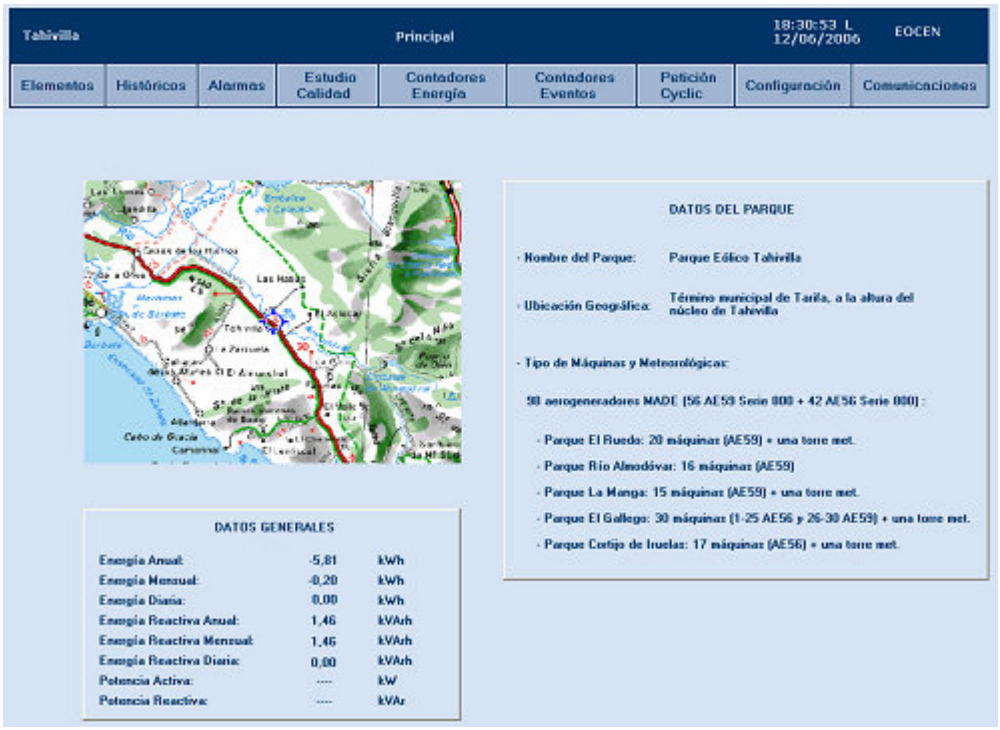


Imagen 19: Pantalla principal

3.2.3. Elementos

Al seleccionar esta pestaña, se carga la página *elem_datos.jsp*, que permite monitorizar los valores actuales de los parámetros eléctricos trifásicos de los distintos aerogeneradores que componen un parque o subparque eólico.

La página posee dos partes. En la parte superior se muestran los datos enviados por el ARP instalado en el aerogenerador seleccionado (tensiones, potencias, energías, THD y frecuencia), junto con el identificador de la máquina. En la parte inferior se muestran los datos recogidos por el ADS asociado a dicha máquina, siempre y cuando este exista y se haya configurado el driver de comunicaciones para que solicite dichos valores.

Para la carga de todos estos datos y su actualización cada 5 segundos se utilizó el motor de actualización de datos y el Servlet *CargarDatosElemento.* Este servlet obtiene los datos de las tablas de *eocen_arp* de la base de datos

Cuando se accede a la página por primera vez, no hay ninguna máquina seleccionada, por lo que los casilleros que muestran la información permanecen vacíos. Para poder seleccionar una máquina, en la parte inferior derecha se insertó un botón que da acceso a un menú que permite seleccionar, a través de un pop up, un subparque y dentro de este subparque, un

aerogenerador. Al elegir el aerogenerador actual, mediante JavaScript forzamos el refresco de la página pasándole el identificador del aerogenerador por la URL, lo que genera la carga de los datos del aerogenerador en la página.

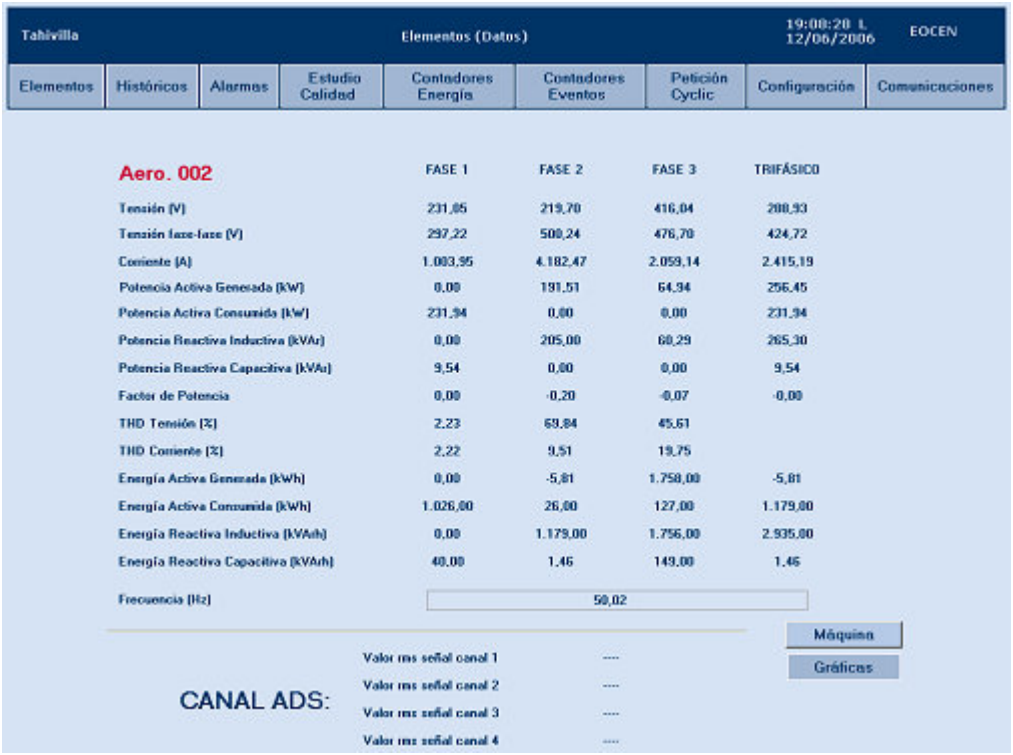


Imagen 20: Pantalla elementos datos

Los pop up que implementan los menús de elección de máquina y subparque son *elige_aero.jsp* y *elige_subparque.jsp* y en ambos casos consisten en un menú desplegable cuyo contenido es obtenido de los datos de subparque del Bean *Sistema*.

Imagen 21: Detalle del menú de selección de subparque

La página posee, además, un botón de gráficas que da acceso a un nuevo menú implementado mediante capas (de igual manera que si hizo con la cabecera). Este menú da acceso a las gráficas de formas de onda y espectros de las tensiones e intensidades tres fases y de los canales ADS.

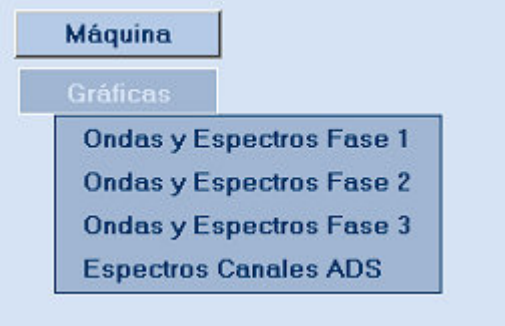


Imagen 22: Detalle del menú de selección de gráficas

Al seleccionar sobre cualquiera de las opciones, se nos redirige a una nueva página, *elem_ondas.jsp* o *elem_ondaads.jsp,* en función de la opción:

- **elem_ondas.jsp**: página compuesta por cuatro gráficas que representan la tensión y la intensidad de la fase seleccionada frente al tiempo junto con sus correspondientes espectros.

  La generación de las gráficas se realiza mediante los Servlets del motor de gráficas: *GraficasBarrasEocen*, para las gráficas de espectros, y *GraficasPuntosEocen*, para las formas de ondas, ya que estas son enviadas por el ARP en forma de vectores de 128 muestras.

  La reutilización de esta pantalla para mostrar los datos de cualquiera de las fases se consiguió discriminando la fase mediante el paso de un parámetro a través de la URL. En función de este parámetro se solicita a la clase *ConIniGraficas* que cargue la configuración correspondiente a una fase u otra en el Bean *VarGraficas.*

  Los datos mostrados en las gráficas se obtienen de la última trama enviada por el ARP de la máquina correspondiente, y se actualizan cada 20 segundos. La actualización se lleva a cabo de manera secuencial, para evitar dejar la pantalla sin datos, actualizando mediante JavaScript la URL de la etiqueta <img>, lo que provoca que se vuelva a iniciar la ejecución del Servlet correspondiente. Para evitar que el navegador cachee

dicha URL, y por lo tanto no actualice la imagen, se introduce en la URL un nuevo parámetro, *time,* que representa la fecha actual en milisegundos, de manera que cada vez que se solicita la actualización de la imagen, este parámetro cambia. La adición de este parámetro no afecta en nada al funcionamiento del motor de gráficas.

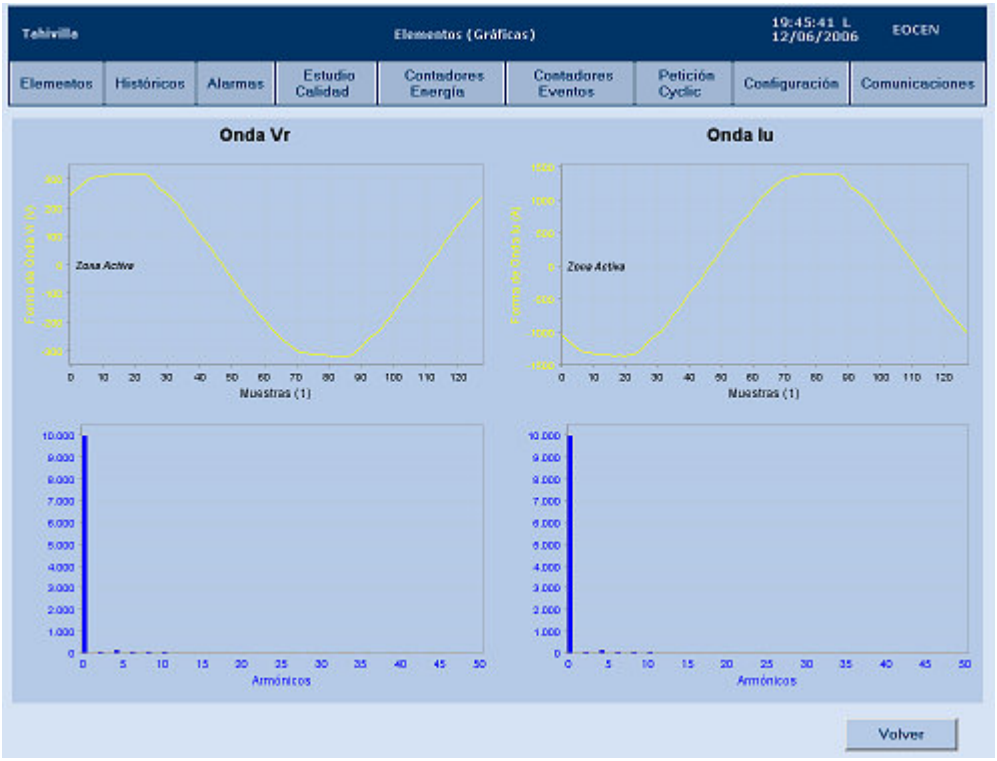


Imagen 23: Pantalla elementos de onda

Otro problema que se presentó a la hora de implementar la actualización de forma secuencia fue cómo almacenar los parámetros de configuración de cada una de las gráficas, ya que cada vez que se llama al Servlet, los datos del Bean *VarGraficas* se actualizan. La solución adoptada consistió en utilizar 4 instancias distintas de dicho Bean, una por cada gráfica.

Finalmente, se añadió un botón para permitir volver a la página *elem_datos.jsp.*

- **elem_ondaads.jsp**. esta página es esencialmente igual a la anterior, con la única diferencia que las 4 gráficas representan espectros, los correspondientes a cada uno de los canales ADS, por lo que la generación de las gráficas se hace a través del Servlet *GraficasBarrasEocen* del motor de gráficas.

3.2.4. Históricos

La pestaña de históricos permite el acceso a un conjunto de páginas destinadas a la monitorización de los parámetros históricos del sistema almacenados en la base de datos.

Al hacer clic sobre esta pestaña, aparece un submenú que permite la seleccionar entre datos, gráficas y datos de producción. Cada una de las opciones nos da acceso a una nueva página:

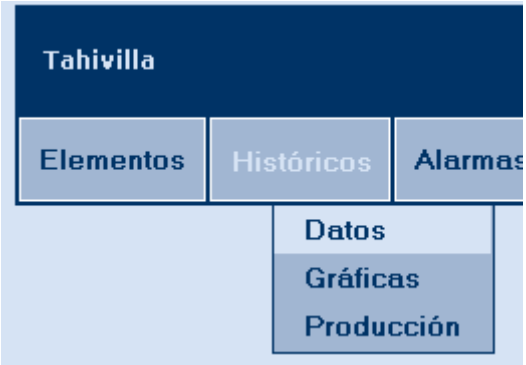


Imagen 24: Detalle del submenú de históricos

- **historicos_datos.jsp**: esta página se utiliza para la consulta de los históricos de las variables trifásicas enviadas por los ARP y de los 4 canales controlados por el ADS. Es idéntica en cuanto a estructuración y variables mostradas a *elem_datos*, con la única diferencia de que incluye un menú que permite seleccionar la fecha, hora y máquina de los datos que queremos visualizar.

  Este menú se implementó a través de un formulario con los siguientes campos:

  - Aerogenerador: desplegable cuyos elementos se obtienen del Sistema indicándole el parque eólico actual contenido en el Repositorio.

  - Hora, minutos y segundos: menús desplegables para seleccionar el instante de los datos que queremos consultar. Por defecto viene preseleccionada las 00:00:00 horas.

  - Fecha: cadena con el formato dd/mm/yyyy que por defecto presenta la fecha actual. Para permitir trabajar de una manera más cómoda se utilizaron unas funciones JavaScript disponibles en Internet que, al pasar el puntero del ratón por encima del campo fecha, muestran un calendario. Al seleccionar una fecha

sobre el calendario, el propio código JavaScript se encarga de rellenar este campo con la fecha en el formato correcto. Todas estás funciones se encuentran en los ficheros *calendar.js*, *calendar-es.js* y *calendar-setup.js*.

Imagen 25: Detalle del calendario para la selección de fechas de la clase *calendar.js*

Cuando rellenamos los campos del formulario y pulsamos el botón “Ver Histórico”, mediante JavaScript se verifica que los valores introducidos son correctos y se realiza una llamada al Servlet *Historico,* contenido en el paquete *scweocen.datos.*

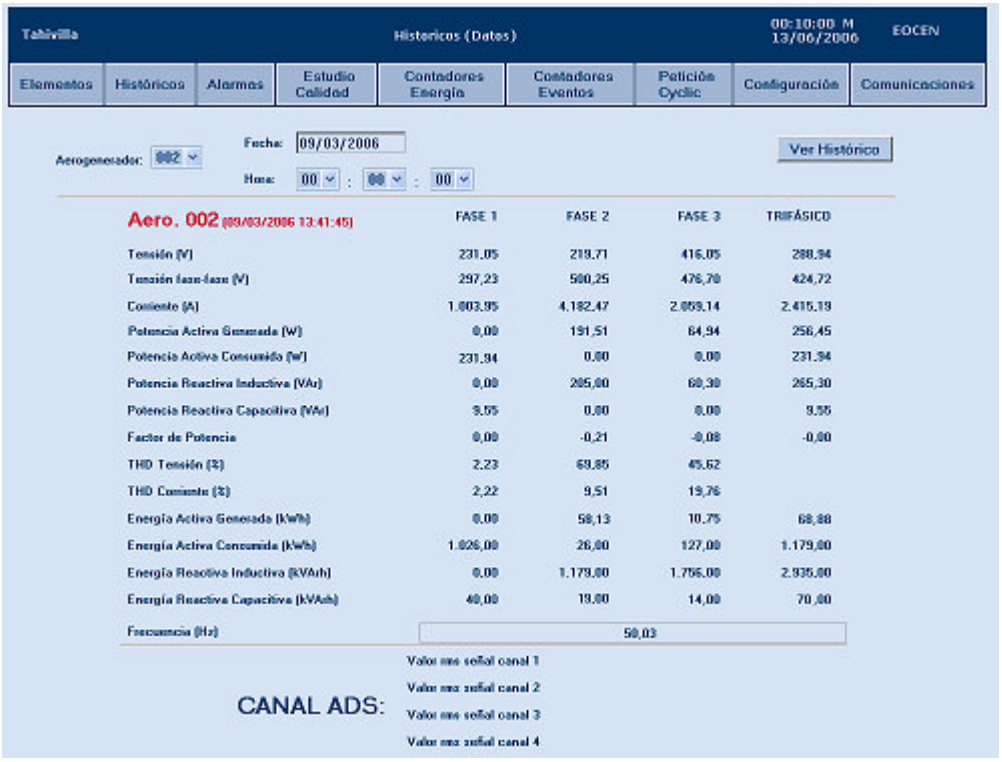


Imagen 26: Pantalla históricos de datos

Este Servlet construye la consulta SQL para recuperar los datos y la ejecuta. Caso de existir los datos solicitados, se introduce el objeto *ObjBD* resultante de la consulta junto con la fecha, hora y máquina en el contexto de sesión. En caso contrario, se sondea la base de datos en busca de los registros inmediatamente anterior y posterior a la hora seleccionada y se introducen los campos *fecha_hora* de dichos registros en el contexto de sesión. Una vez finalizado el proceso, se redirige la petición a la misma página, con lo que se fuerza el refresco de la misma.

Al iniciarse la carga de la página, se revisa si hay algún dato en el contexto de sesión, pudiendo darse tres situaciones: que no haya ninguno, debido a que es la primera vez que se accede a la página, en cuyo caso los menús presentan los valores por defecto y no se muestra ninguna información por pantalla; que haya datos que se corresponden con los parámetros de búsqueda introducidos previamente en el menú, ante lo cual el menú presenta la fecha, hora y máquinas seleccionadas y en la pantalla se muestra la información contenida en el contexto de sesión; o bien que se haya realizado una consulta para la cual no hay datos, por lo que los elementos que tenemos en el contexto de sesión son los campos *fecha_hora* de los registros inmediatamente previo y posterior encontrados en la base de datos. En este último caso, mediante funciones JavaScript, se deshabilita el menú, no se muestra ninguna información y se presenta la posibilidad de seleccionar uno de los dos instantes localizados. Al realizar la selección, se repite el mismo proceso exactamente igual que si hubiéramos introducido los datos a mano en el formulario.

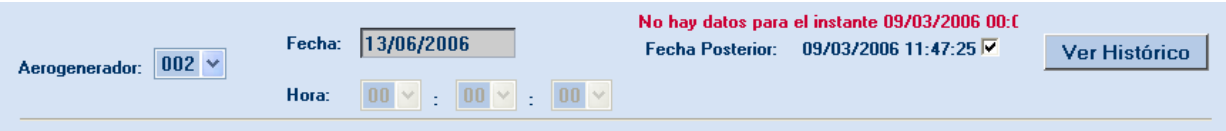


Imagen 27: Detalle del menú de la página de históricos de datos cuando no hay datos para la fecha seleccionada

- **hist_producción:** página diseñada para consultar los resúmenes acumulativos horarios, diarios, mensuales y anuales de energías y factores de potencia de todo el parque eólico.

  Presenta un menú que permite seleccionar la fecha, haciendo de nuevo uso de los calendarios JavaScript, y la hora sobre las que queremos obtener la información. Este menú, al igual que ocurría en *hist_datos*, es un formulario que al activarse clickeando en el botón “Ver”, llama al Servlet *HistoricosProducción*.

La misión de este Servlet es, dada una fecha y hora, realizar 4 consultas a la base de datos para solicitar las variables a mostrar en la página sobre periodos distintos: desde el inicio hasta el final de la hora seleccionada (resumen horario), desde las 00:00 horas del día hasta la hora seleccionada (resumen diario), desde el día 1 del mes hasta el la fecha y hora seleccionada (resumen mensual) y desde el 1 de enero del año hasta la fecha y hora seleccionada (resumen anual). Tras insertar los resultados en un objeto *TreeMap* de nombre *datosProd* y depositar este en el contexto de sesión, el Servlet finaliza redirigiendo la petición a la misma página, forzando así su refresco.

Al iniciarse la carga de la página, se sondea el contexto de sesión en busca del mapa *datosProd.* Si este existe y contiene datos, se muestran por pantalla. En caso contrario, no se muestra ninguna información.

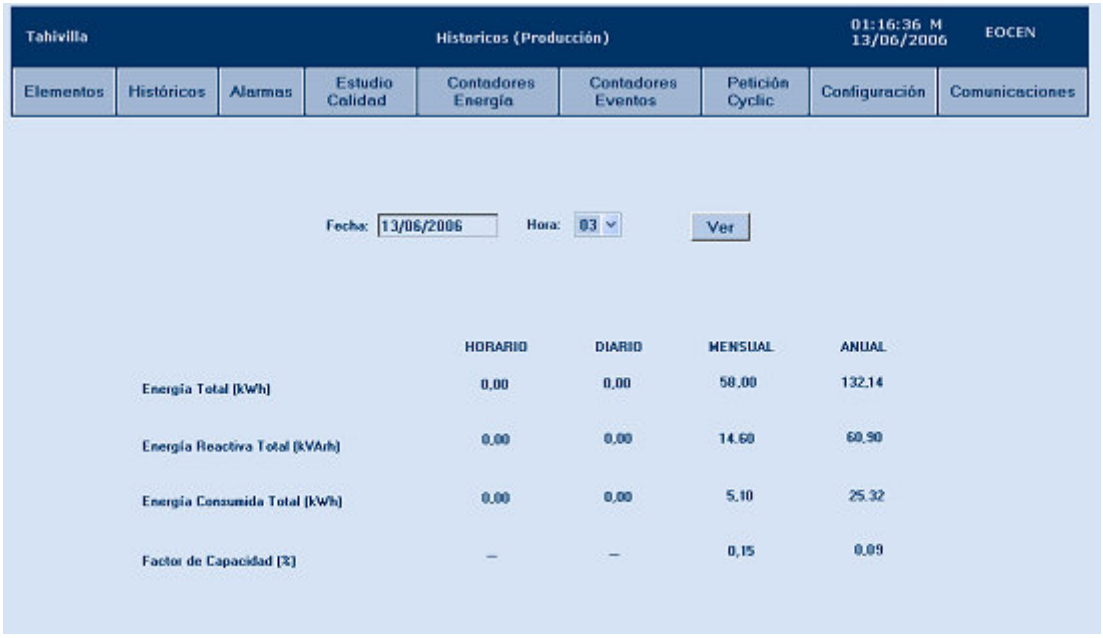


Imagen 28: Pantalla de históricos de producción

- **hist_graficas**: esta pantalla se diseñó para permitir monitorizar la evolución de los parámetros eléctricos a través de gráficas.

  La pantalla está formada por un menú de tres solapas implementado con funciones JavaScript y un iframe cuyo contenido varía en función de la pestaña seleccionada:

  - Formas de onda: Permite visualizar simultáneamente una o varias formas de onda de cualquiera de las 3 tensiones o intensidades trifásicas.

- Espectros: Muestra el espectro de una de las intensidades, tensiones o de un canal ADS, según se elija.

- Tendencias: Muestra la evolución de las magnitudes seleccionadas durante la hora previa a la fecha y hora seleccionadas. Permite seleccionar entre las distintas tensiones, intensidades, potencias, energías y THD.

La página que se carga en el iframe, *iframe_hist.jsp,* consta de un menú que permite elegir la fecha, hora y máquina cuyos datos queremos consultar y una gráfica que inicialmente se carga vacía.

Al seleccionar una pestaña, el cambio del contenido del iframe se consigue cambiando la URL del mismo, que seguirá apuntando a la *iframe_hist.jsp*, pero con un parámetro "tipoMenu" distinto. En función de este último, se carga en el iframe un menú de variables u otro a través de directivas JSP *<include>.* Los distintos menús de variables disponibles se encuentran en los archivos *menuGraficasOndas.jsp*, que presenta 6 checkbox que permiten seleccionar una o varias tensiones e intensidades trifásicas; *menuGraficasEspectros.jsp*, con 8 radiobuttons para elegir un espectro de entre las intensidades, tensiones y canales ADS; y *menuGraficasTendencias.jsp*, que contiene un menú desplegable para elegir el tipo de variables de entre tensiones, intensidades, THD, potencias y energías. En este último caso, al seleccionar un grupo, se fuerza el refresco del iframe, mostrándose un conjunto de checkbox para la selección de uno o varios de las variables del grupo seleccionado.

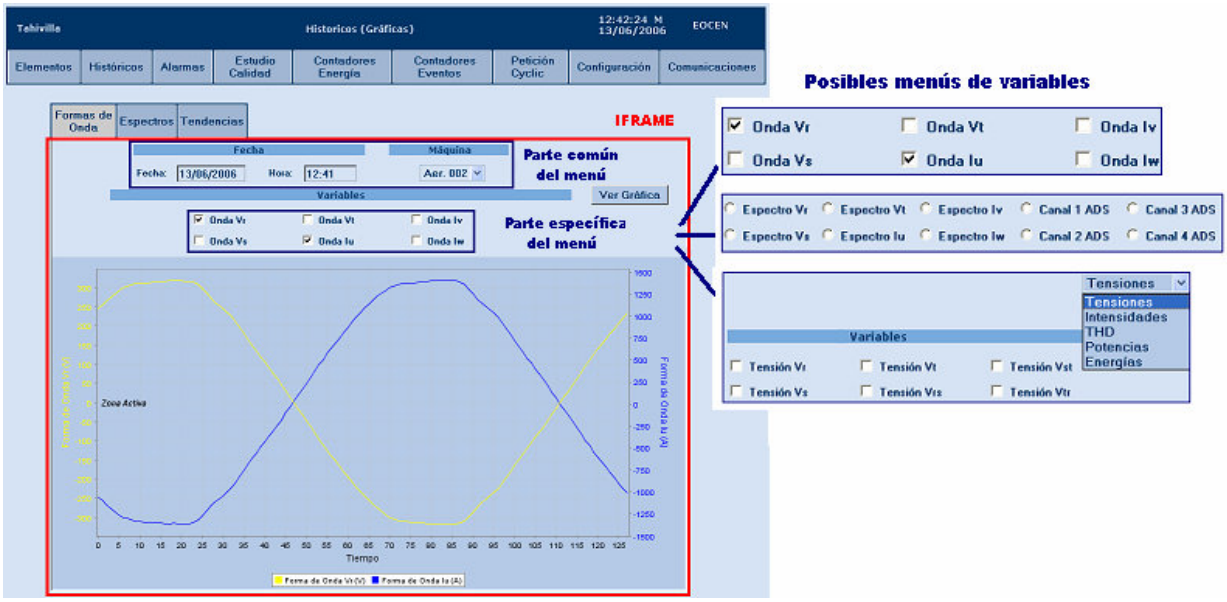


Imagen 29: Pantalla de históricos de gráficas y los posibles menús de variables

A diferencia de todos los casos vistos hasta ahora, para la generación de las gráficas, en vez de utilizar el Bean *VarGraficas*, se recurre al uso de funciones JavaScript, que recorren la página recogiendo la información sobre las variables seleccionadas y generan a partir de ellas la URL necesaria para llamar al Servlet del motor de gráficas y posteriormente la insertan en la etiqueta <*img*> del iframe.

3.2.5. Alarmas

La pestaña de alarmas da acceso a dos páginas que permiten monitorizar las alarmas en tiempo real o bien los históricos de alarmas producidas en un intervalo de tiempo seleccionado.

- **Alarmas en tiempo real**: esta opción permite supervisar las alarmas que se van produciendo en tiempo seudo-real. Aunque teóricamente deberían refrescarse en tiempo real, los retrasos de transmisión y procesado de los datos de alarmas hacen que sea inviable, por lo que se estableció un tiempo de refresco de 15 segundos. Teóricamente, el sistema sería capaz de trabajar con tiempos de refresco inferiores a 5 segundos; la elección de 15 segundos como tiempo de refresco se hizo para dar un margen de seguridad y evitar una carga excesiva de trabajo al servidor.

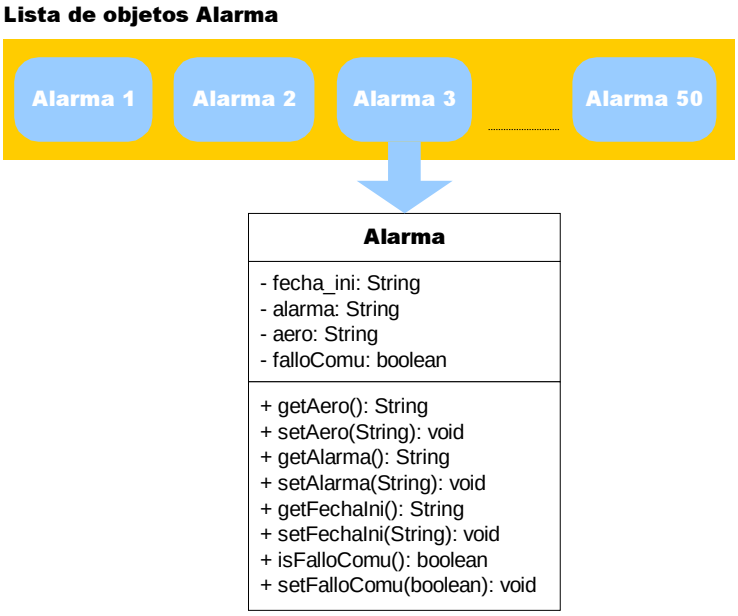


Imagen 30: Almacenamiento y estructura de los objetos *Alarma*

Al seleccionar esta opción en el menú de alarmas, se realiza una llamada al Servlet *RealHistoricos* del paquete *scweocen.datos.* La misión de este Servlet consiste en recuperar de la base de datos las últimas 50 alarmas almacenadas (las más recientes). Los datos de cada alarma se introducen en un objeto tipo Alarma (*scweocen.beans*), que no es más que un Bean con atributos *fecha_ini*, *aero*, *id_alarma* y *falloComunicación*, con los correspondientes métodos *set* y *get* de cada uno de ellos.

A continuación, se introducen los objetos Alarma correspondientes a cada alarma en un objeto List que se deposita en el contexto de sesión, y se redirige la petición a la página *alarm_hist.jsp*, que se limita a recuperar la información de las alarmas del contexto de sesión, formatear los datos mediante el uso de las funciones de la clase *Formateo*, y mostrarlos por pantalla.

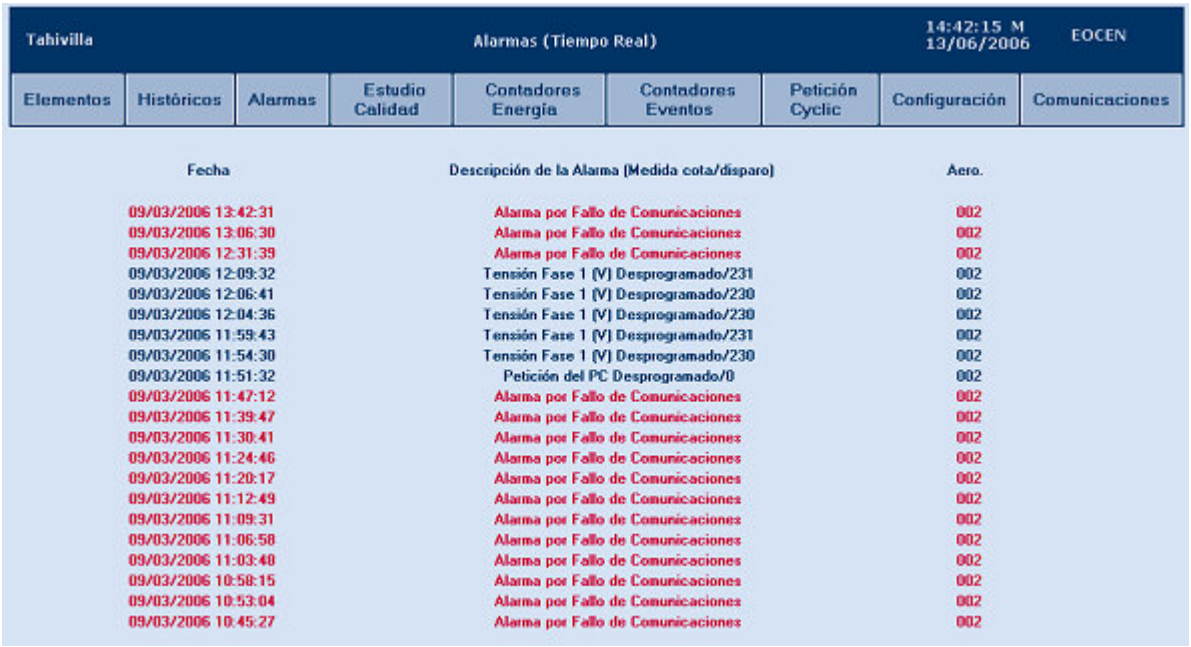


Imagen 31: Pantalla de alarmas en tiempo real

La actualización de la lista de alarmas se consigue programando en la etiqueta <*body*> de la página una llamada al Servlet a través de la URL de la página cada 15 segundos, llamada que pone de nuevo en marcha el mecanismo de recuperación y presentación de las alarmas.

- **Alarmas históricos**: pantalla que permite recuperar los históricos de un determinado tipo de alarmas producidas en el intervalo temporal especificado.

  Esta pantalla esta formada por un menú que permite especificar las fechas de inicio y fin del periodo de consulta, campos que se implementaron usando nuevamente los calendarios JavaScript de la clase *calendar.js,* el tipo de alarma y el aerogenerador. Estos dos últimos campos se presentan mediante un menú desplegable cuyos ítems se obtienen del Bean Sistema, con la ayuda del Repositorio para saber cuál es el parque actual y poder obtener la lista de los aerogeneradores.

  Además, para evitar tener que refrescar toda la página cada vez que se solicita una consulta nueva, se insertó un iframe en el que se carga la página *iframe_alarmas.jsp*, que es la encargada de mostrar los datos.

  Al hacer clic en “Ver Históricos”, se recogen y validan los datos del formulario y se lanza la ejecución del Servlet *HistoricoAlarmas* desde el iframe. Esto se consigue generando en el iframe con JavaScript un formulario oculto con los campos necesarios para albergar la información recogida y realizando la llamada al Servlet desde dicho formulario del iframe.

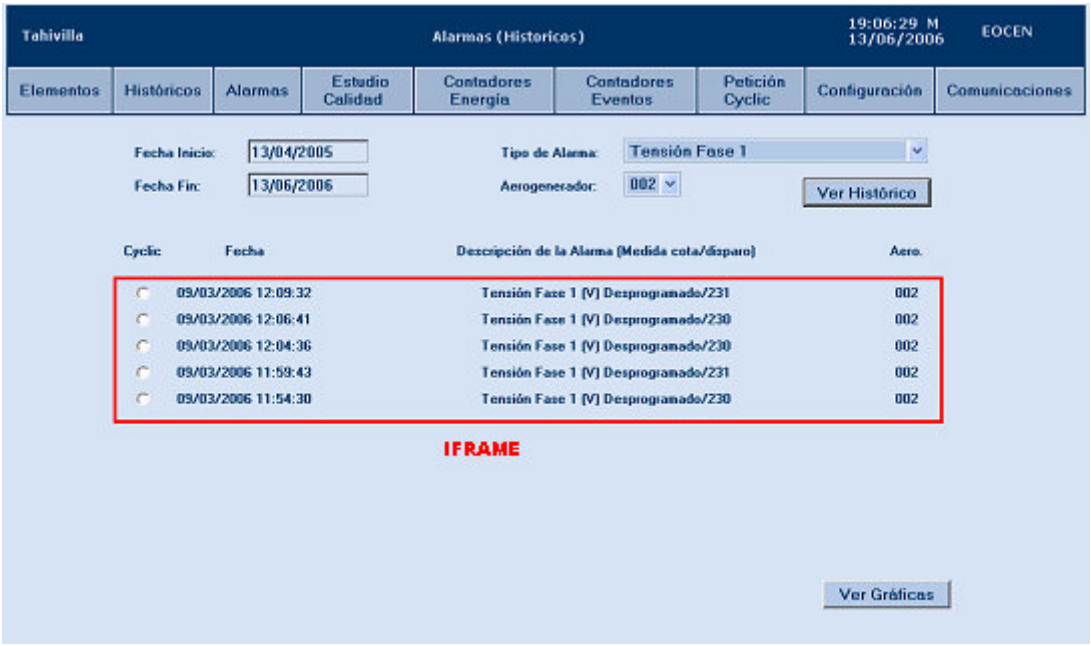


Imagen 32: Pantalla de históricos de alarmas

El Servlet se encarga de construir la consulta para obtener las alarmas seleccionadas y almacenan los resultados en una lista de objetos *Alarma*, que posteriormente deposita en el contexto de sesión antes de redirigir la petición a *iframe_alarmas.jsp*.

Finalmente, el iframe se encarga de sacar la lista de alarmas del contexto de sesión y mostrar la información de las mismas por pantalla.

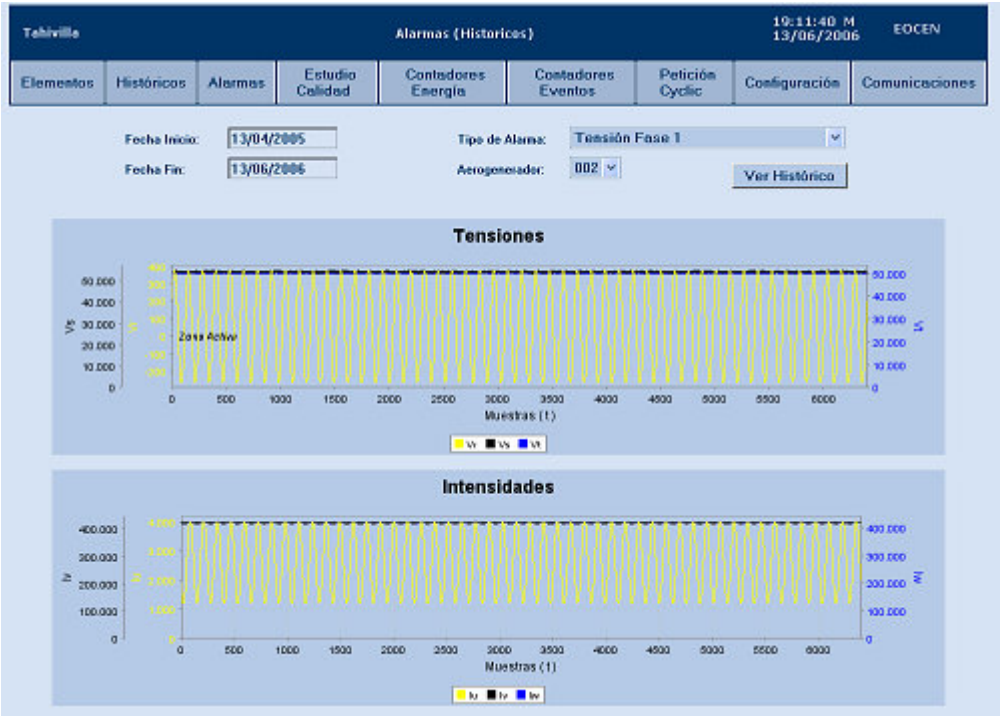


Imagen 33: Visualización de los cyclics asociados a una alarma en la pantalla de históricos de alarmas

La última funcionalidad que aporta la página es la posibilidad de, para aquellas alarmas que llevan asociado un cyclic, mostrarlo por pantalla. Para ello, se añade junto con la información de la alarma un radiobutton que permite seleccionarla y un botón “Ver Gráfica”. Al seleccionar una alarma y pulsar el botón, se carga en el iframe la página *graficasCyclics.jsp,* pasándole como parámetros la fecha de la alarma y el número de la máquina. Esta página carga en dos Beans *VarGraficas* las configuraciones por defecto 91 (gráfica que muestra Vr, Vs y Vt) y 92 (muestra Iu, Iv e Iw) y la fecha pasada como parámetro y realiza la llamada a los Servlets del motor de gráficas.

3.2.6. Estudio de calidad

La página de estudios de calidad permite analizar los resultados obtenidos en dichos estudios y compararlos con los objetivos marcados por la normativa del RD 436/2004.

Para el funcionamiento de esta página, se desarrollo un nuevo Bean, *BeanCalidad (scweocen.beans)*, que contiene un array con los valores máximos impuestos por la norma para cada armónico, que se recogen de las tablas de *eocen_conf* de la base de datos al llamar al constructor del Bean, y una lista con los estudios de calidad. Esta lista estará compuesta por todos los estudios que haya para una máquina, año y mes determinado, parámetros que quedarán recogidos en otros tantos atributos del Bean.

Con la finalidad de facilitar el almacenaje de los datos de cada estudio, se diseñó una clase contenedor, *ObjetoEstudio* (*scweocen.beans*), que contiene un campo fecha, 3 arrays de 50 floats para almacenar los valores de los armónicos de Vr, Vs y Vt y otros 3 arrays de 50 x 15 floats para guardar los valores de los ínter armónicos de cada armónico de cada una de las tres tensiones. Para poder recuperar cada uno de estos valores, se introdujeron tres funciones *getInterArmonicoVx*(numArmonico, numInterArmonico, decimales), donde “x” puede ser r, s o t según la fase de la que queramos la información, que devuelven el ínter armónico del armónico indicados formateados con el número de decimales solicitados y, para el acceso a los datos de los armónicos*, getArmonicoVx*(numArnonico, decimales).

Los métodos públicos que ofrece el *BeanCalidad* son:

- ***getEstudios(maquina, dia, mes, year)****:* su misión consiste en devolver los estudios existentes para la máquina, día, mes y año especificados. Para ello, comienza comprobando si los estudios que tienen almacenados en la lista son del mes, año y máquina seleccionados. En caso de no ser así, accede a la base de datos, obtiene todos los estudios que cumplan esos tres requisitos, almacenando los datos de cada uno en un *ObjetoEstudio*, borra la lista interna antigua y genera una nueva con todos los *ObjetoEstudio* obtenidos. A continuación filtra la lista por el día seleccionado y devuelve las fechas de los estudios resultantes del filtrado como cadenas formateadas en campos <inputs> con el siguiente formato:

```
<option value="yyyy-mm-dd hh :mm :ss ">dd/mm/yy hh :mm</option>
```

- ***getNorma(armonico, decimales***): devuelve el valor máximo marcado por la norma para el armónico seleccionado con tantos decimales como se especifique en el segundo argumento de la función.

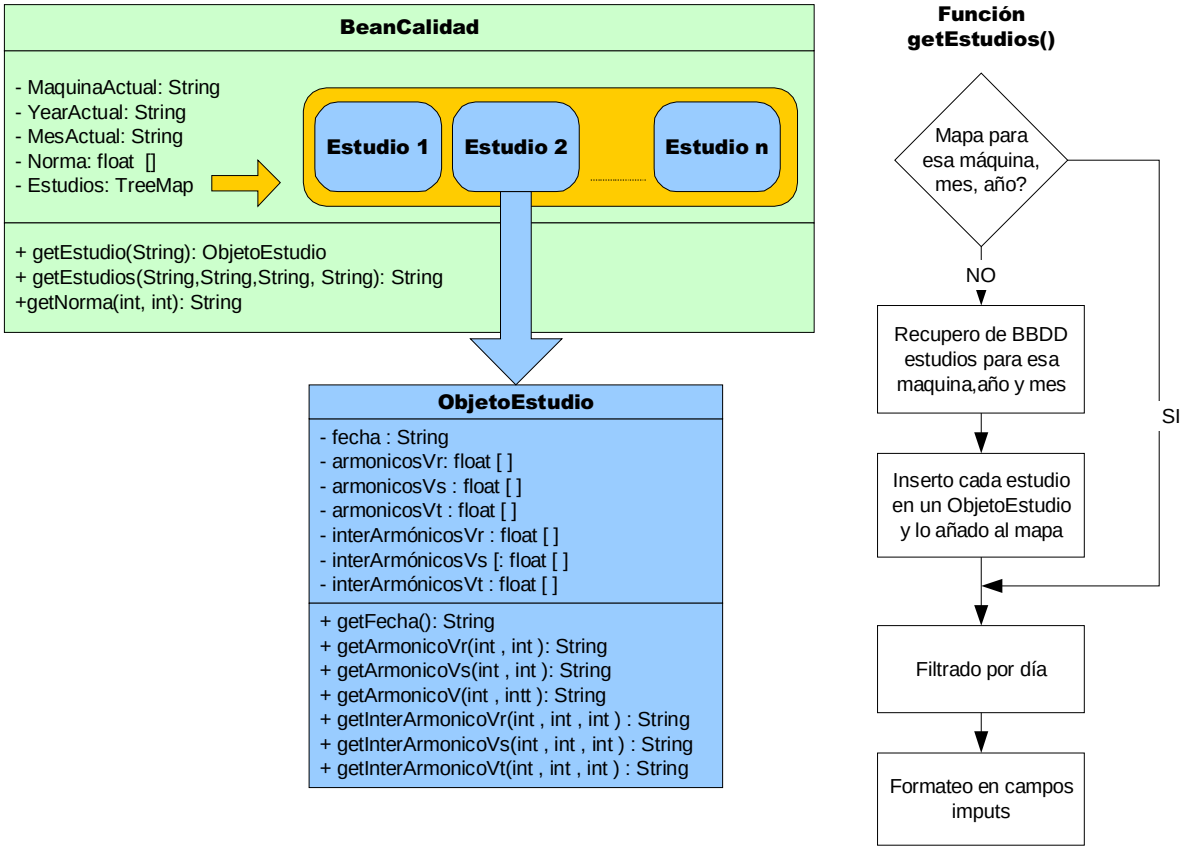


Imagen 34: Estructura de almacenamiento de los datos de los estudios de calidad en el *BeanCalidad* y diagrama de flujo del método *getEstudios()*

- ***getEstudio(String fecha):*** función que busca en la lista interna de estudios si hay alguno que se corresponda con la fecha indicada y caso de que existiera, devuelve el *ObjetoEstudio* que contienen los datos de dicho estudio.

Una vez implementadas las herramientas necesarias para el funcionamiento de la página, se realizó el diseño de la misma, dividiéndola en tres partes funcionales:

- **Menú de filtrado de datos**: permite la selección del día, mes, año, máquina e intervalo diezminutal del estudio de calidad que deseamos visualizar. Este menú se implementó sobre un iframe en el que se carga la página *menuCalidad.jsp*, constituida por un formulario con una lista desplegable por cada uno de los campos anteriormente descritos. Los campos de todas estas listas, a excepción del de los intervalos diezminutales, se rellenan mediante funciones JavaScript, con la ayuda del Sistema y

del Repositorio en el caso de los aerogeneradores. La lista de periodos diezminutales para los que hay estudio se obtiene mediante la función *getEstudios()* del *BeanCaldidad*, que devuelve todos los intervalos ya formateados como campos inputs para ser introducidos directamente dentro del formulario.

La actualización de la lista de estudios disponibles se realiza al modificar alguno de los parámetros de filtrado enviando los datos del formulario, acción que fuerza la recarga del iframe.

El formulario contiene además de los campos anteriormente descritos un botón que permite actualizar el resto de la página una vez seleccionado un estudio.

- **Tabla de comparación de los valores de los armónicos con la norma**: este campo está constituido por un iframe en el que se carga la página *datosCalidad.jsp,* que contiene una tabla en la que se presentan los valores de los percentiles 95% de los 50 armónicos de cada una de las tensiones trifásicas, junto con el valor máximo permitido por la norma para cada uno de ellos.

  Inicialmente, este iframe permanece vació, hasta que seleccionamos un estudio y pulsamos el botón “Aplicar” del menú de filtrado. Este botón origina la recarga del iframe, durante la cual se extraen los datos del estudio y de la norma mediante los métodos *getEstudio()* y *getNorma()* del *BeanCalidad*. Una vez finalizada la carga de datos, se recorren los valores que toman las tres tensiones en cada percentil y se comparan con la norma. Caso de que alguno de ellos sobrepasase la norma, se cambia el color del dato a rojo para resaltarlo mediante JavaScript.

- **Gráficas de los percentiles:** La pantalla contiene dos iframes más, *graficasCalidad1.jsp*, que muestra los valores de los percentiles de una de las tensiones; y *graficasCalidad2.jsp*, que muestra los máximos de entre los percentiles las tres tensiones frente a los valores dictados por el RD.

  Inicialmente, ambas gráficas están vacías, hasta que se selecciona un estudio y se pulsa sobre el botón “Aplicar”. Este botón, genera el refresco de los dos iframes, indicándoles el identificador de la gráfica que debe cargar cada uno (el 61 en el caso de *graficasCalidad2 y el 71, que se corresponde con los percentiles de Vr, en el caso de graficasCalidad1.* La generación de estas gráficas se realiza mediante el motor de gráficas cargando en VarGraficas las configuraciones predeterminadas para los identificadores especificados.

Para permitir cambiar en la primera gráfica entre los percentiles de Vr, Vs y Vt, se incluyeron tres botones en la página cuya misión es forzar el refresco del iframe que contiene esta gráfica cambiándole el identificador de la gráfica (71 para Vr, 72 para Vs y 73 para Vt).

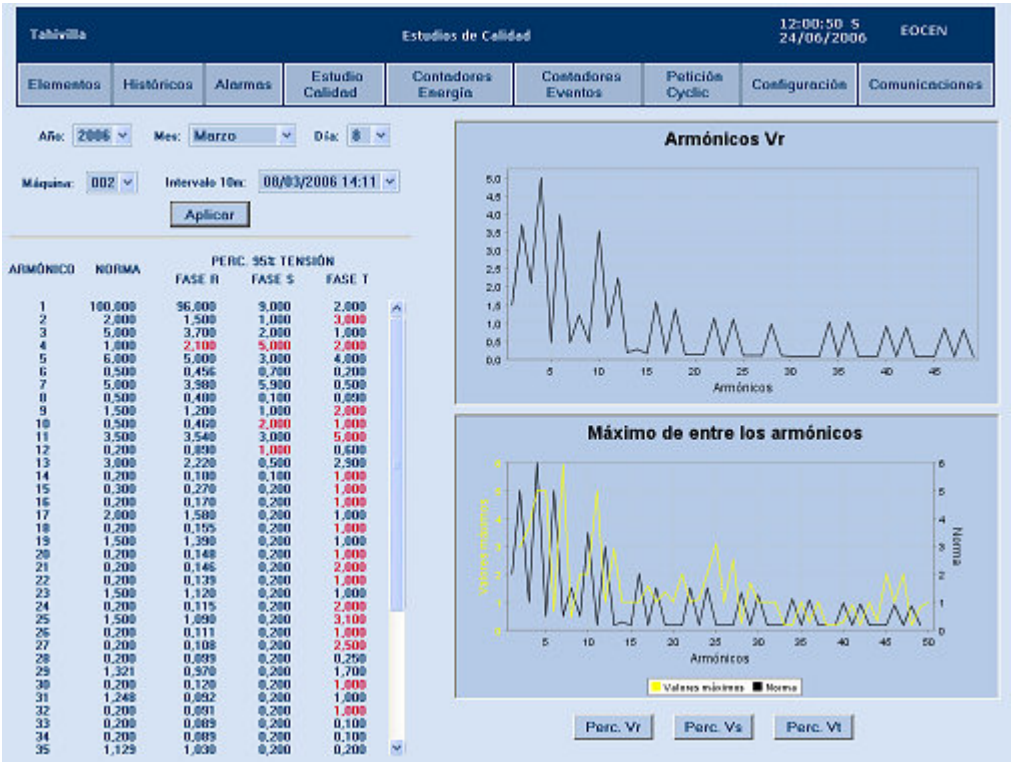


Imagen 35: Página de estudios de calidad

Finalmente, para permitir analizar esta gráficas en detalle, en las etiquetas <img> de cada una se introdujo un evento *onClick*, de tal manera que al pinchar sobre una de las gráficas, se abra un popup (*popupCalidad1.jsp* y *poupCalidad2.jsp*) a tamaño página completa que tiene el mismo contenido que los iframes de las gráficas pero cambiando la configuración el tamaño de la gráfica para que esta aparezca ampliada.

3.2.7. Contadores de energía

Pagina que permite monitorizar y resetear los contadores acumulativos de energía de cada uno de los aerogeneradores de un parque.

La parte superior de la pantalla contiene una tabla con la energía activa generada trifásica actual y mensual, la energía reactiva mensual (suma de las energías reactiva inductiva

trifásica y reactiva capacitiva trifásica), y la energía trifásica consumida de cada una de las máquinas del parque. La actualización de estos datos cada 5 segundos se lleva a cabo mediante el motor de refresco de datos usando el Servlet *CargarDatosEnergias,* que obtiene los valores de las tablas de *eocen_arp* contenidas en la base de datos.

La parte inferior de la pantalla muestra un cuadro de controles que permite el reseteo de los contadores de energía de cada máquina. Este cuadro dispone de un desplegable para elegir la máquina deseada mas dos botones que permiten, mediante funciones JavaScript seleccionar todas las máquinas del parque o limpiar la selección hecha, y un tercer botón que se encarga de generar y transmitir la orden haciendo uso del motor de envío de órdenes, al que le pasa el número de la máquina y el identificador 1, correspondiente a la orden de reseteo de contadores. Esta orden no requiere de parámetros, por lo que en el lugar de estos se enviarán al módulo dos arrays vacíos.

Imagen 36: Página de contadores de eventos

3.2.8. Contadores de eventos

Esta página se diseñó para permitir al operador consultar el número de alarmas de cada tipo que se han producido en un intervalo de tiempo.

La página consta de un menú que permite filtrar la fecha, hora, máquina y tipo de eventos que queremos consultar.

- **Tipo de eventos**: la clasificación de los eventos se hace según el parámetro eléctrico que los hace desencadenarse. Para seleccionar los eventos, la página consta de un conjunto de checkbox, que se generan obteniendo los parámetros eléctricos que pueden hacer saltar el evento mediante la función *obtenerMedidas()* del Bean *Sistema*.

  Para facilitar el trabajo de selección, se dotó a la página de dos botones que, a partir de funciones JavaScript, permiten seleccionar todos los tipos de eventos o deseleccionar los previamente elegidos.

- **Fecha y hora**: la fecha se selecciona mediante los calendarios suministrados por la clase *calendar.js*. Para la hora se dispone de un menú desplegable

- **Aerogenerador**: Se elige a través de un menú desplegable cuyos elementos se obtienen sacando del Bean *Sistema* los aerogeneradores que posee el parque actual, que se obtiene del Repositorio.

El funcionamiento de la página requiere de un elemento más, el botón ”Ver Contadores”. Al pulsar dicho botón, mediante funciones JavaScript, se crea un formulario oculto en la misma página con los campos aerogenerador, fecha, hora, número de eventos seleccionados y un array con los identificadores de los eventos seleccionados. Al terminar de generarse el formulario, se realiza un *submit* del mismo, que se traduce en una petición al Servlet *ContadorEventos (scweocen.datos).*

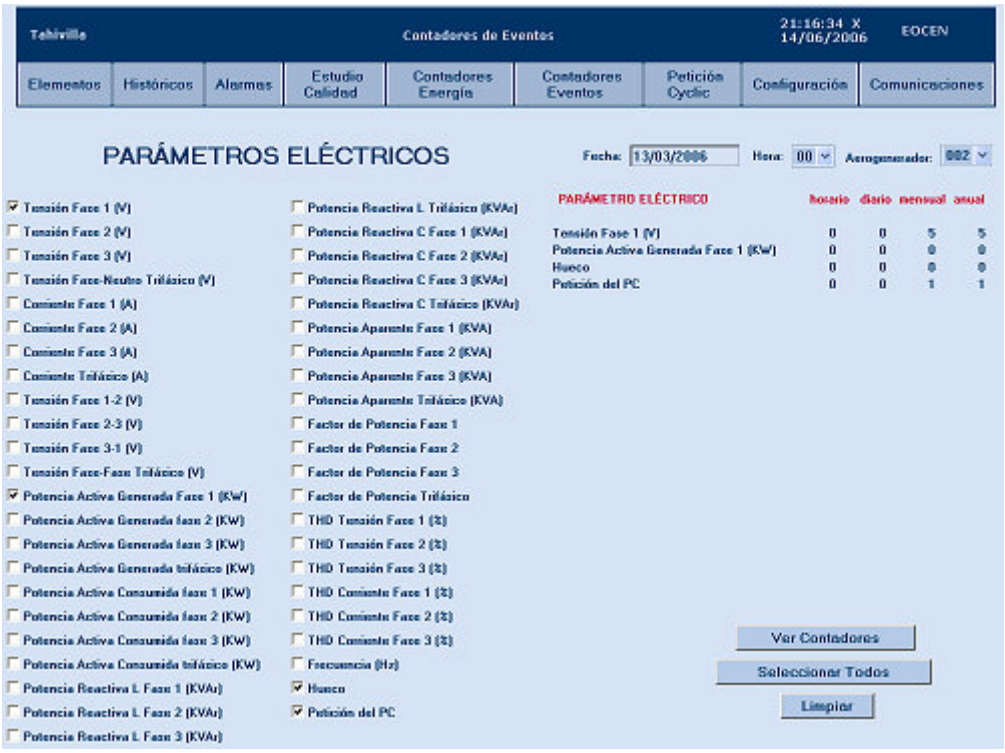


Imagen 37: Pantalla de contadores de eventos

El Servlet se encarga de realizar cuatro consultas a las tablas de *eocen_alarmas* para obtener el número de veces que se ha producido cada uno de los eventos requeridos en la hora previa a la seleccionada, el día previo al seleccionado, los últimos 30 días y los últimos 12 meses respectivamente. Con los resultados obtenidos, genera un objeto *TreeMap* en el que las claves son cada una de las causas de los eventos consultadas y el valor almacenado es un nuevo objeto *TreeMap* de clave el periodo (“H”,”D”,”M” y ”A”) y el valor se corresponde con el número de alarmas de ese tipo que se han producido en dicho intervalo temporal.

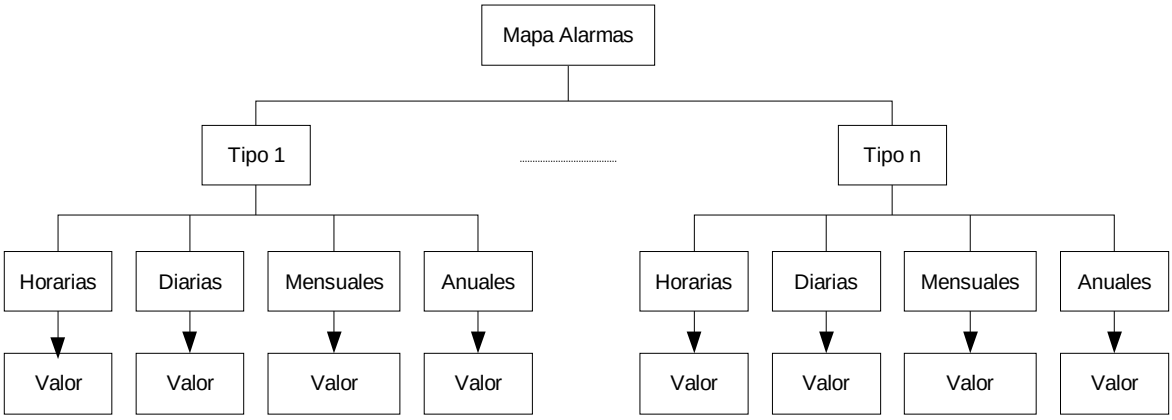


Imagen 38: Estructura del mapa en el que se almacenan las alarmas recuperadas

Finalmente, el Servlet introduce el mapa junto con el identificador del aerogenerador consultado y la fecha y hora en el *Request* de la petición y redirige la solicitud nuevamente a la página. Esta, al recargarse, saca del objeto *Request* el mapa, junto con el resto de parámetros de la consulta, formatea su contenido y lo muestra por pantalla.

3.2.9. Petición de cyclic

La función de esta pantalla consiste en permitir al operador solicitar, sin que se haya producido ninguna alarma, el envió de un cyclic por parte de cualquiera de los ARP instalados en el sistema, así como facilitar el análisis de los cyclics recibidos por este método a través de gráficas de las tres tensiones e intensidades.

La pantalla consta de dos partes. La primera consiste en un cuadro que permite recuperar de la base de datos los cyclics almacenados que se hayan producido por una petición por parte del operador. Para filtrar los estudios a recuperar, se introdujo un menú que permite especificar la fecha de inicio y final del intervalo temporal en el que estamos interesados a través de dos calendarios, implementados mediante la clase *calendar.js,* y la máquina, a través de un desplegable cuyos campos se obtienen del *Sistema* especificándole el parque actual contenido en el Repositorio. El menú se completa con un botón “Recuperar” y un iframe en el que se carga la página *iframe_cyclic.jsp.*

Cuando pulsamos el botón “Recuperar”, se realiza una petición al Servlet *Cyclics* (*scweocen.datos*), que se encarga de acceder a las tablas del esquema *eocen_alarmas* de la base de datos para recuperar todas aquellas que se hayan producido en el intervalo especificado y que tengan como causa 50 (código correspondiente a las peticiones de cyclic por parte del operador). Los resultados obtenidos los formatea en forma de campos <*input*> para poder introducirlos directamente en el desplegable del iframe que nos permite elegir el estudio y los almacena en el contexto de sesión. Finalmente, redirige la petición al iframe, que se recarga, recuperando la cadena generada por el Servlet del contexto de sesión e introduciéndolo en el código de la página dentro de las etiquetas <*select*>.

Para poder pedir nuevos cyclics a los ARP, se introdujo un botón “Solicitar Cyclics”. Al pinchar sobre el, se cargan los mismos pop up con los menús para elegir el subparque y dentro de él la máquina a la que queremos enviar la orden que ya utilizamos en la página principal. Una vez seleccionada la máquina, se pide la conformidad al operador para mandar la

petición, tras lo cual, utilizando el motor de envío de órdenes, se construye y envía una petición con identificador 2 (petición de cyclic) y sin parámetros.

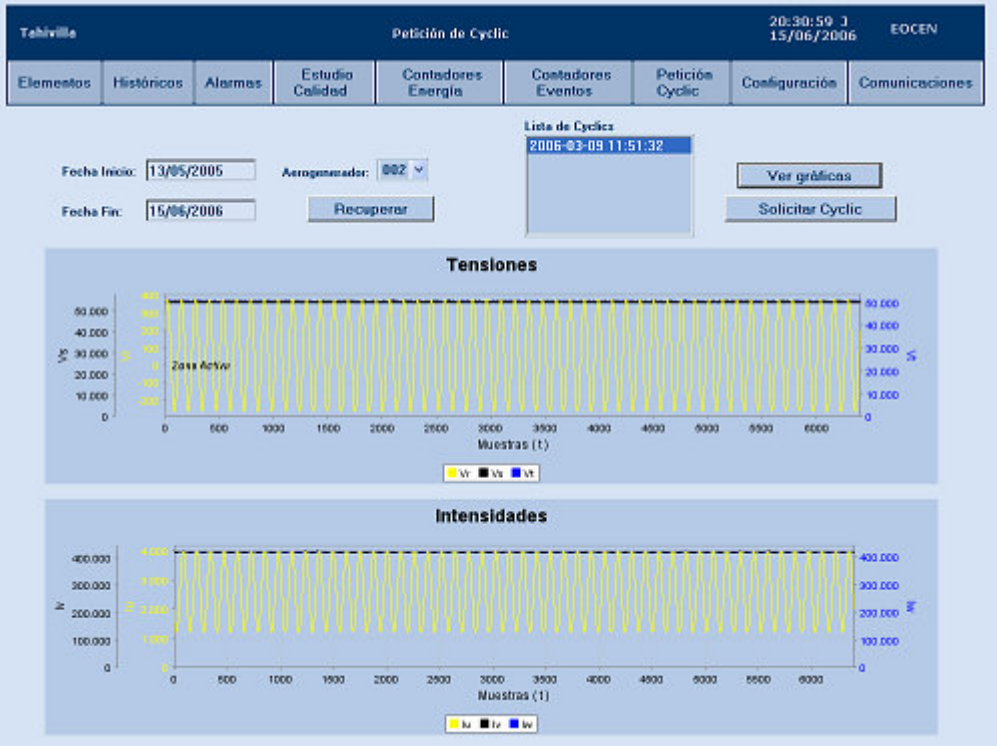


Imagen 39: Pantalla de petición de cyclics

Como ya explicamos en el proceso de obtención y descomposición de un cyclic, desde que se realiza la petición hasta que el cyclic está disponible en la base de datos puede pasar un tiempo superior a un minuto, por lo que es necesario sondear continuamente la base de datos en busca de nuevos cyclics. Para ello, en la etiqueta *<body>* de la página se programó una llamada al Servlet cada 15 segundos, llamada que desencadena todo el proceso de actualización del iframe.

La segunda parte de la página esta formada por un iframe en el que se reutiliza la página *graficasCyclics.jsp* desarrollada para la página de históricos de alarmas, que inicialmente muestra dos gráficas vacías, y un botón “Ver Gráficas”. Al seleccionar un estudio y pulsar sobre el botón mediante código JavaScript se fuerza el refresco del iframe pasándole como parámetro por la URL la fecha del cyclic seleccionado. La página del iframe, al cargarse llama al motor de generación de gráficas para que devuelva dos gráficas, la primera de ellas con la representación de las tres tensiones trifásicas recogidas en el cyclic y la segunda con las tres intensidades.

3.2.10. Configuración

Esta pestaña da acceso a la pantalla que permite la configuración de los eventos programables por el operador. Los eventos programables son disparadores basados en condiciones sobre variables físicas del sistema que pueden ser programadas por el operador. Cuando se dispara un evento programable, salta una alarma y el ARP de la máquina en la que se ha producido envía un cyclic.

Inicialmente, por convenio, al arrancar el sistema no habrá ningún evento configurado, y por lo tanto las tablas de *eocen_conf.conf_configuraciones* en la que se guarda la información sobre los mismos estarán vacías. Será responsabilidad del operador establecer, mediante las funcionalidades aportadas por esta página, la configuración inicial deseada para cada máquina.

Para el almacenamiento de la configuración de los eventos se desarrolló una clase contenedora, *ObjetoConfiguración* (*scweocen.beans*), con los atributos necesarios para guardar toda la información referente a la configuración de cada uno de los tres posibles eventos de una máquina: ARP en el que se han configurado, índice, parámetro eléctrico, cota y sentido del disparador, periodo del día y días de la semana durante los que el disparador está activado. Todos estos atributos tienen sus correspondientes métodos *get* y *set*.

El método constructor de esta clase, al ser invocado, se encarga de acceder a la tabla *eocen_conf.conf_configuraciones* de la base de datos para obtener y almacenar en sus atributos los datos de la configuración de la máquina que se le pasa como parámetro.

Los elementos *ObjetoConfiguración* de cada una de las máquinas se almacenan a su vez en un Bean, *BeanConfiguración* (*scweocen.beans*), que contiene como único atributo un mapa en el que las claves son los identificadores de la máquina y los valores los correspondientes *ObjetoConfiguración.* En este Bean se definió una función pública *imprimeConfiguracion(máquina),* que tomando como parámetro el identificador de la máquina, devuelve la información sobre la misma ya formateada con las etiquetas HTML necesarias para ser introducidas directamente en una página Web. Caso de no encontrarse en el mapa interno el *ObjetoConfiguración* de la máquina solicitada, se llama al constructor de *ObjetoConfiguración* pasándole el identificador de la máquina. Una vez que este recupera los datos de dicho ARP, se inserta el objeto resultante en el mapa y se formatea la información que contiene.

Para permitir actualizar la información del Bean, se definió una segunda función, *borra*(maquina), que tomando como parámetro el identificador del ARP, busca en la lista el *ObjetoConfiguracion* correspondiente a dicha máquina y, si existe, lo borra, con lo que en la próxima solicitud de información de dicha máquina se fuerza un acceso a la base de datos para recuperar los valores actualizados.

La página de configuración de eventos programables está dividida en 2 partes. La primera de ellas permite consultar los eventos configurados para cada una de las máquinas. Consta de un desplegable, cuyos campos se obtienen del Sistema indicándole el parque actual contenido en el Repositorio, que permite seleccionar el aerogenerador y un iframe, *iframe_eventos.jsp*, en el que se muestra la información sobre cada uno de los 3 posibles eventos configurados.

Para cargar la información de una determinada máquina, tas seleccionarla y pulsar sobre el botón “Cargar” se fuerza una recarga del iframe refrescando su URL, a la que se le añade el número de la máquina. El iframe, al cargarse, llama al método *imprimeConfiguracion()* del Bean e inserta en su código la cadena que éste devuelve con la información ya formateada.

Imagen 40: Pantalla de configuración de eventos programables. Muestra el momento en que se pide la conformidad antes de enviar una configuración nueva con los parámetros elegidos

La segunda parte contiene un menú que permite configurar los eventos. Este menú consta de los siguientes campos:

- **Aerogenerador**: desplegable que permite elegir la máquina
- **Número de evento**. todas las máquinas pueden tener hasta 3 eventos configurables que se identifican con los números del 1 al 3
- **Horas de inicio y fin**: permiten marcar el periodo temporal durante el que el disparador estará activo.
- **Días de la semana en que estará habilitado**: checkbox que permite habilitar el evento en uno o varios días de la semana. También está la opción de deshabilitar
- **Parámetro eléctrico**: magnitud física sobre la que se impondrá la condición del disparador. Se elige a través de un desplegable cuyos campos se obtienen del objeto *ObjBD* devuelto por el método *obtenerMedidas()* del Bean *Sistema*.
- **Cota**: valor del parámetro eléctrico que hace saltar el evento
- **Sentido**: indica si el evento salta al superar la cota por arriba o por abajo

El menú se completa con un botón “Aplicar”, que al ser pulsado y tras comprobar mediante funciones JavaScript que todos los campos del formulario se han rellenado correctamente, recoge los parámetros introducidos y realiza una llamada al motor de envío de órdenes para que envíe una orden con *id* 6 (configuración de un evento programable). Este tipo de órdenes requieren de un total de 14 parámetros:

Identificador corto *	Parámetro	Posibles valores
_PET_CF_E0@_DOMI	Disparador habilitado el Domingo	0 (falso), 1 (verdadero)
_PET_CF_E0@_LUNE	Disparador habilitado el Lunes	0 (falso), 1 (verdadero)
_PET_CF_E0@_MART	Disparador habilitado el Martes	0 (falso), 1 (verdadero)
_PET_CF_E0@_XCOL	Disparador habilitado el Miércoles	0 (falso), 1 (verdadero)
_PET_CF_E0@_JUEV	Disparador habilitado el Jueves	0 (falso), 1 (verdadero)
_PET_XX_XXX_PAR1	Número de evento a configurar	1,2 ó 3
_PET_CF_E0@_VIER	Disparador habilitado el Viernes	0 (falso), 1 (verdadero)
_PET_CF_E0@_SABA	Disparador habilitado el Sábado	0 (falso), 1 (verdadero)
_PET_CF_E0@_INIH	Hora de inicio de funcionamiento del disparador	Hora con formato hh:mm:ss
_PET_CF_E0@_FINH	Hora de fin de funcionamiento	Hora con formato hh:mm:ss

_PET_CF_E0@_COTA	Cota del parámetro eléctrico que hace saltar el evento	Numérico
_PET_CF_E0@_INDI	Índice que identifica la variable eléctrica sobre la que se aplica la cota (ver Anexo C)	Cualquiera de los índices (ver tabla eocen_conf.def_medidas
_PET_CF_E0@_SCOM	Sentido de la comparación	0 (negativo), 1 (positivo)
_PET_CF_E0@_HABI	Disparador habilitado	0 (falso), 1 (verdadero)

Tabla 2: Lista de parámetros requeridos para enviar una orden de configuración de un evento programable.

- El formato completo de los nombres de los parámetros es E_AN_TR_TV_NNN_ seguido del identificador corto, donde “E” es el identificador del sistema (en este caso España), “AN” el de la región (Andalucía), “TR” el del nodo (Tahivilla), “TV” el del parque eólico (Tahivilla) y “NNN” el número de la máquina. El carácter arroba del identificador corto se sustituye por 1,2 ó 3, según el número del parámetro que estemos configurando.

El módulo de órdenes, a la hora de procesar una petición, comprueba si la orden a enviar corresponde con la configuración de un evento programable. En caso de ser así, tras enviar todos los parámetros y el identificador de la orden, espera a recibir la trama de asentimiento por parte del ARP y cuando esta llega, actualiza la información previa sobre el evento configurado en la tabla *eocen_conf.conf_configuraciones*. Para actualizar la información en el *BeanConfiguración,* ejecuta la función *borrar()* ,especificándole la máquina, con lo que se consigue eliminar de la lista interna del Bean el *ObjetoConfiguración* correspondiente a esa máquina, por lo que la próxima vez que se solicite la información sobre los eventos de dicha máquina, al no encontrar el objeto en la lista, el Bean deberá acceder a la base de datos para recuperar la información.

3.2.11. Comunicaciones

Esta pantalla no contiene funcionalidad alguna y se limita a mostrar una imagen que presenta la estructura organizativa de los sistemas de comunicaciones instalados para dar cobertura al sistema.

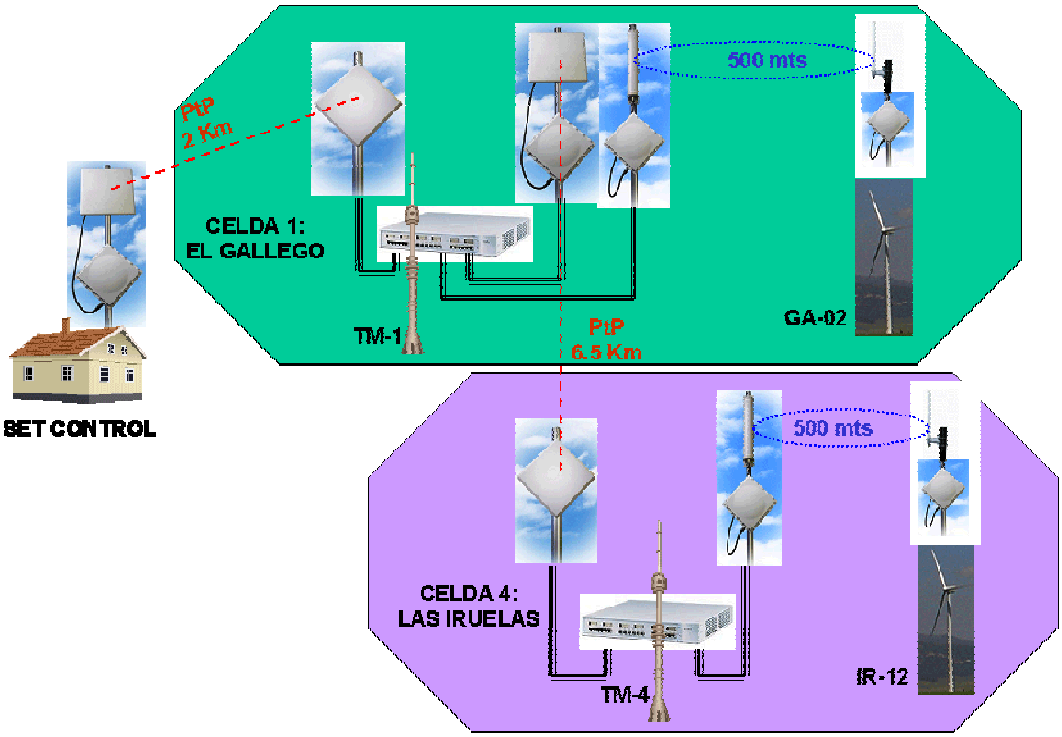


Imagen 41: Esquema de comunicaciones presentado en la página de comunicaciones

3.3. Instalación y configuración del servidor Tomcat []

Como ya se comentó en las memorias descriptiva y justificativa de este proyecto, el sistema de visualización esta desarrollado con tecnologías Web de fuentes abiertas, siendo la principal de ellas JSP, que requiere para su funcionamiento de la presencia de un servidor Apache Tomcat, que es un contenedor de aplicaciones JSP que permitirá el acceso al SCWEOCEN desde cualquier ordenador conectado a la intranet del parque, y de la versión adecuada de JDK (Java Development Kit), que contiene la máquina virtual Java y las herramientas de compilación necesarias para Java.

La instalación del software se hizo de forma remota, debido a las complicaciones de acceso al servidor principal, enracado en un armario. Para la transmisión de ficheros y ejecución de comandos se hizo uso de la herramienta SSH.

El proceso seguido para instalar el Tomcat fue:

- Verificar si el servidor está dotado de la versión adecuada de la JDK, la 1.5, en el directorio */usr/local/java*. En este caso, el servidor principal de la aplicación la traía ya preinstalada junto con el sistema operativo Ubuntu, por lo que no hizo falta instalarla.

- Descargar el instalable comprimido de la página del proyecto Apache-Tomcat: http://tomcat.apache.org/. La versión utilizada es la 5.5.16, la más reciente que había en el momento de realizar la instalación.

- Descomprimir el archivo descargado y moverlo a la carpeta */usr/local/tomcat5*, que no existía, por lo que hubo que crearlo. Para ello usamos los siguientes comandos:

```
mkdir /usr/local/tomcat5
tar –xvzf apache-tomcat-5.5.17.tar.gz
mv carpetaTomcatCreada  /usr/local/
```

- Crear el script de arranque del servicio. Éste se debe localizar en el directorio */etc/init.d* y llamarse tomcat5. Para que el script pueda funcionar, debemos incluir en el mismo el JAVA_HOME. El código del script es el siguiente:

```
#!/bin/sh
#
# Startup script for Tomcat
#

export CATALINA_HOME=/usr/local/tomcat
export JAVA_HOME=/usr/local/java/java

case "$1" in
start)
  echo -n "Starting Tomcat 5"
  echo
  /usr/local/tomcat/bin/startup.sh
  ;;
stop)
  echo -n "Stopping Tomcat 5"
  /usr/local/tomcat/bin/shutdown.sh
  ;;
restart)
  $0 stop
  $0 start
  ;;

*)
  echo "Usage: $0 { start | stop | restart }"
  echo
  exit 1
esac
```

exit 0

Este script requiere permiso de ejecución para todos los usuarios del sistema. Además, como se necesita que el servicio se inicie al arrancar la máquina, se incluyó un enlace simbólico en */etc/rc2.d/* (directorio run level2; el sistema operativo, al arrancar, inicia todos los servicios cuyo script se encuentre en este directorio).

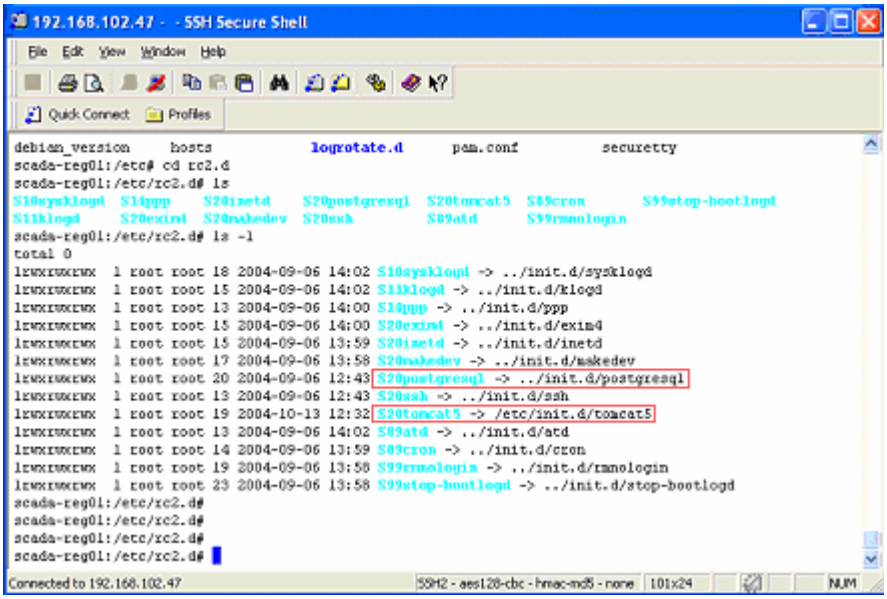


Imagen 42: Contenido del directorio de /etc//rc2.d/. Marcados en rojo los scripts de arranque del Tomcat y de la PostgreSQL

chmod a+x tomcat5
ln –s /etc/init.d/tomcat5 S20tomcat5

- Instalar las librerías XWindows que aportan al sistema operativo las características y herramientas visuales necesarias para soportar tanto el servicio Tomcat como la aplicación SCWEOCEN:

  - o libice6
  - o libsm6
  - o libx11-6
  - o libxext6
  - o libxp6
  - o libxt6

- libxtst6

La instalación de estas librerías se realizó buscando la versión más reciente de cada una de ellas incluida en la distribución Ubuntu con la ayuda de los comandos *apt:*

```
apt-cache search libreríaAbuscar
apt-cache install paquete
```

- Realizar las modificaciones necesarias en el script /usr/local/tomcat/bin/Catalina.sh para que el Tomcat funcione correctamente:
  - Aumentar la memoria disponible hasta un máximo de 400 megas
  - Cargar las librerías AWT al iniciar el Tomcat

Para ello, se incluyeron las siguientes líneas en el script:

```
# Linea para la carga de las awt
echo "Configurando JAVA para funcionar sin pantalla"
CATALINA_OPTS="$CATALINA_OPTS -Djava.awt.headless=true"

# Linea para ampliar la memoria disponible para java
export JAVA_OPTS='-Xms100m -Xmx400m'
```

- Añadir un usuario *admin* con los permisos de administrador que nos permita desplegar y replegar las aplicaciones. Para ello, editamos el fichero XML *tomcat-users* ubicado en */usr/local/tomcat5/conf/* y añadimos las siguientes líneas:

```
<?xml version="1.0" encoding="utf-8" ?>
<tomcat-users>
 <role rolename="tomcat" />
        <role rolename="role1" />
        <role rolename="manager" />
        <role rolename="admin" />
        <user username="tomcat" password="tomcat" roles="tomcat" />
        <user username="role1" password="tomcat" roles="role1" />
        <user username="both" password="tomcat" roles="tomcat,role1" />
        <user username="admin" password="" roles="admin,manager" />
</tomcat-users>
```

3.4. Despliegue del SCWEOCEN en el servidor Tomcat

Una vez finalizada la fase de desarrollo del SCWEOCEN, para que la aplicación pueda estar disponible a los operadores a través del servidor Tomcat, es necesario exportar todos los archivos que lo componen a dicho servidor.

Este proceso se llevó a cabo a través de un archivo war, que es un fichero comprimido que contiene de forma estructurada las clases Java ya compiladas, librerías, imágenes, páginas JSP y HTML y archivos de configuración de la aplicación.

La generación de dicho archivo war se hizo de forma automática a través del Eclipse.

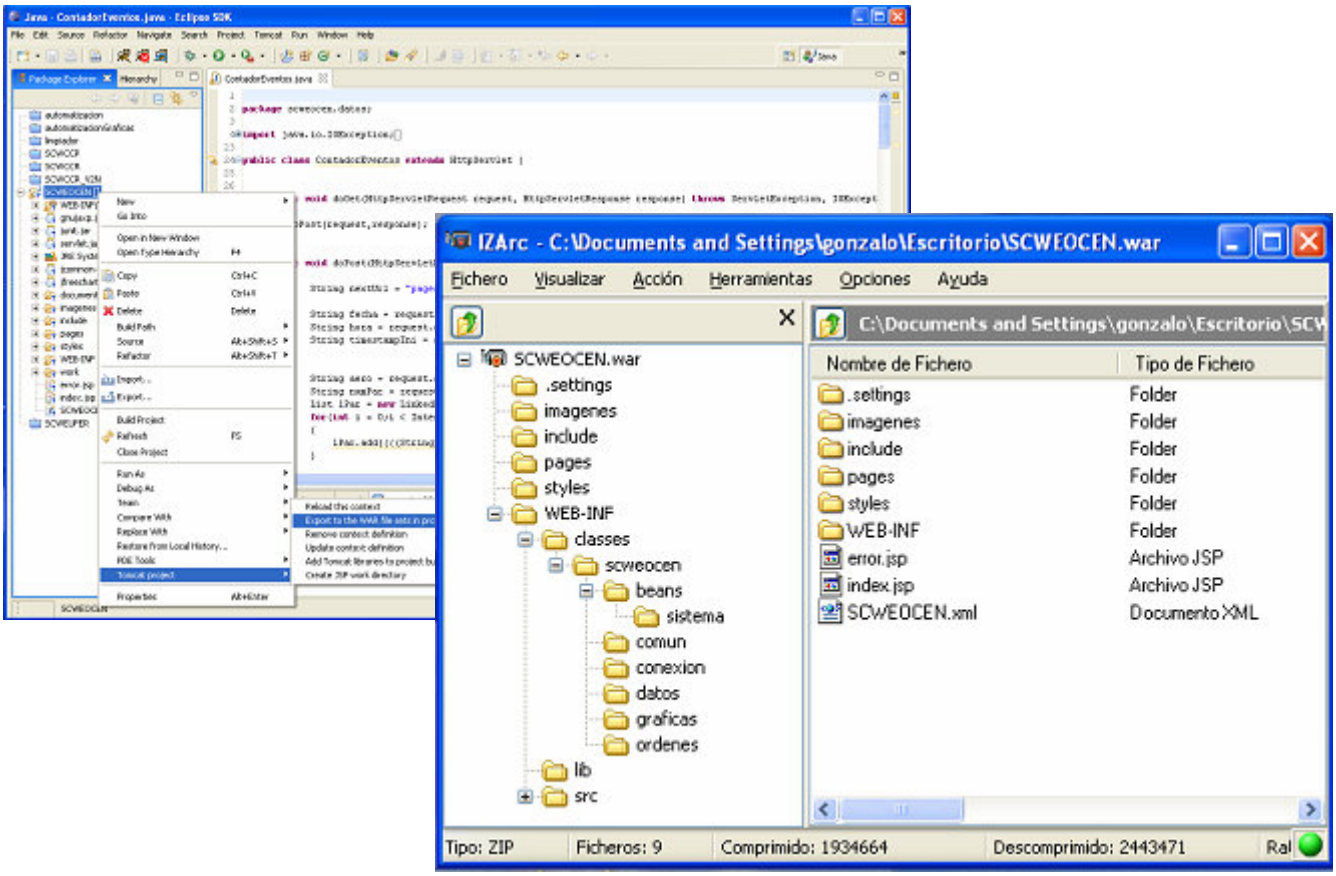


Imagen 43: Generación de un archivo war con el eclipse y estructura del mismo

Una vez generado el war, con el servidor Tomcat arrancado, se accedió a la herramienta de administración de éste para proceder a desplegar la aplicación. Posteriormente, se detuvo el servidor para incluir en el directorio */usr/local/tomcat5/conf/Catalina/loacalhost* el archivo XML de configuración requerido por el Tomcat, SCWEOCEN.xml, y se reinicio el servidor. Esta

parada y reinicio fue necesaria para asegurarnos de que se cargaba dicho fichero de configuración.

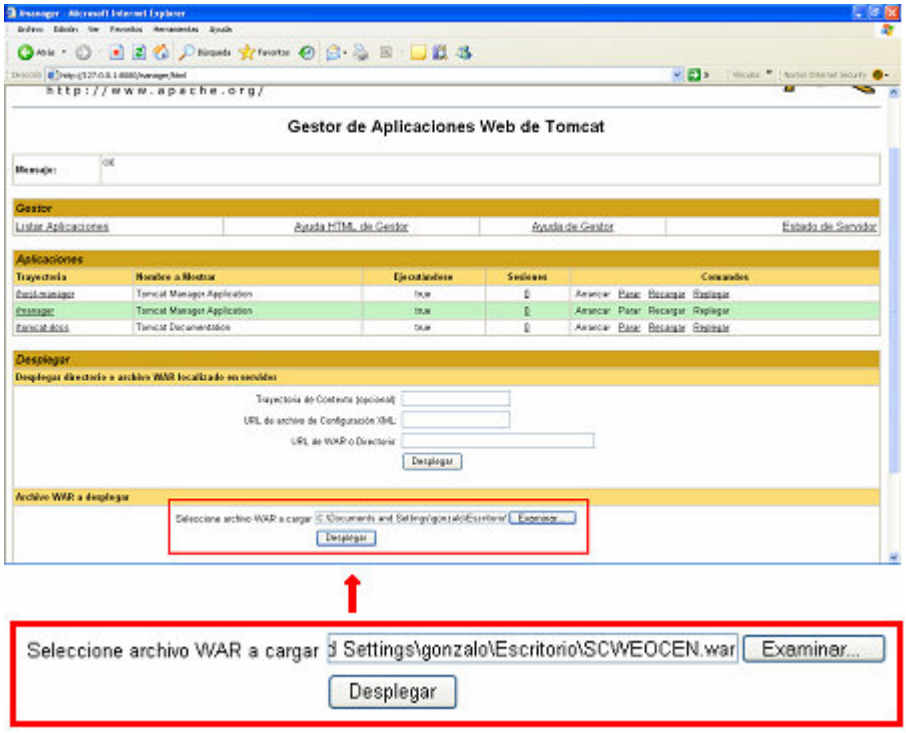


Imagen 44: Despliegue del war

4. Núcleo y plugins

El núcleo es un programa cuya finalidad es dar soporte y organizar el trabajo de los distintos módulos necesarios para dotar al sistema de las funcionalidades requeridas para poder llevar a cabo las tareas de supervisión y control de los parámetros del sistema.

Como tal, no posee funcionalidad alguna por si sólo, ya que éstas están contenidas en los distintos plugins que se le incorporan:

- **Driver de comunicaciones**: plugin encargado de las tareas referentes al intercambio de información con los ARP que componen el sistema. Implementa los protocolos de comunicación desarrollados, presentando una interfaz de comunicaciones totalmente transparente al resto de módulos del sistema.

- **Módulo de bombeo**: aplicación cuya misión es realizar las inserciones periódicas de los datos recibidos de los ARP y ADS o procedentes del módulo de cálculo en las tablas de la base de datos.

- **Módulo de cálculo**: programa encargado de hacer los cálculos necesarios sobre las variables recibidas antes de que estas sean almacenadas. Este módulo es el responsable de realizar los algoritmos estadísticos sobre los estudios de calidad recibidos.

- **Plugin de envío de órdenes**: servidor HTTP encargado de recibir y procesar las tramas generadas por el módulo de envío de órdenes del SCWEOCEN con los parámetros e identificadores de cada orden.

- **Módulo de alarmas**: aplicación cuya misión es realizar un seguimiento de las variables, garantizando el cumplimiento de determinadas condiciones sobre ellas. Caso de violarse alguna condición, genera una alarma, notificándolo al resto de módulos involucrados y guardando un registro en la base de datos

Esta estructura modular del sistema permite incorporar más capacidades añadiendo nuevos plugins que las implementen, sin que esto afecte en el funcionamiento del resto de módulos.

El núcleo se diseñó como un programa multihilo para trabajar sobre plataformas Linux, siendo los plugins implementados por librerías dinámicas. Por ello, el desarrollo de todos estos

programas se realizó en C++, utilizando las herramientas aportadas por el entorno de programación Eclipse y el compilador g++.

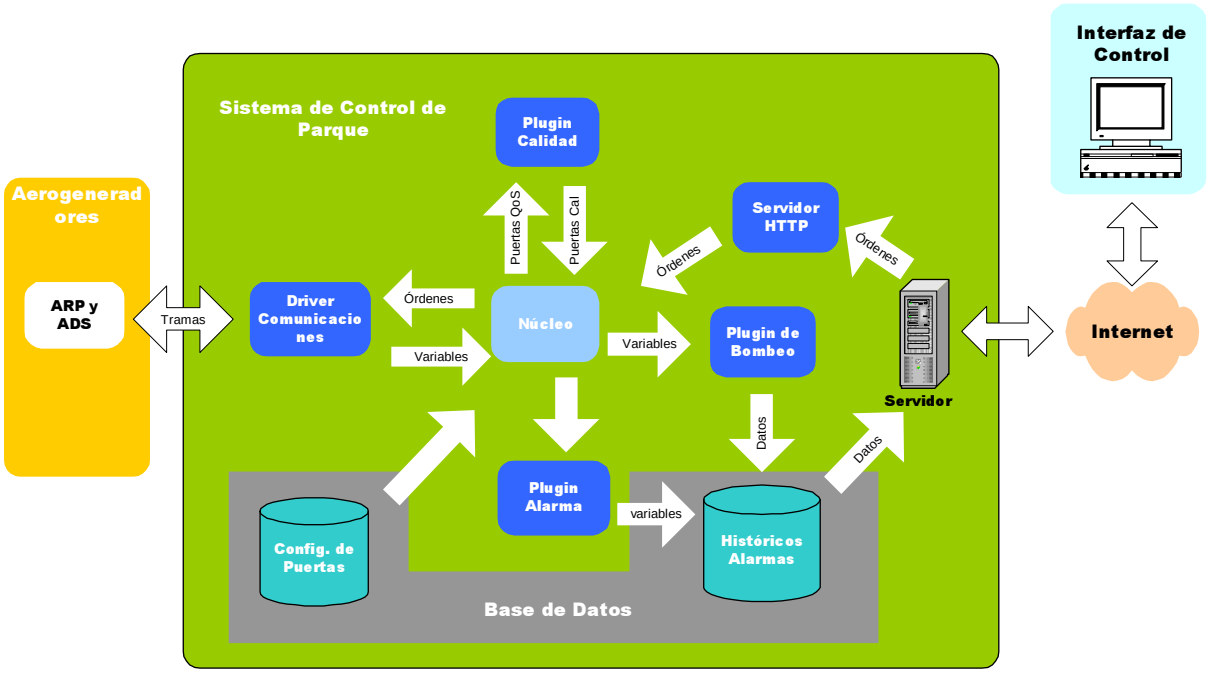


Imagen 45: Esquema general del sistema. En azul oscuro se representan los plugins.

El funcionamiento del núcleo comienza con el arranque del sistema. En el script de arranque, se hace la llamada al programa. Éste, al iniciarse, carga las puertas contenidas en la tabla *eocen_conf.conf_puertas* de la base de datos y arranca a todos los plugins, generando una tabla interna por cada uno de ellos en la que se incluyen las variables a las que se subscriben (aquellas que esperan recibir). De esta forma, cada vez que se recibe una variable, consulta qué plugins están subscrita a ella y se la reenvía.

La configuración de los plugins y el núcleo se hace mediante la inserción de los parámetros requeridos por éstos en las tablas de *eocen_conf* contenidas en la base de datos. Estos parámetros se cargan en los módulos al iniciarse la aplicación, por lo que los cambios que hagamos en dichas tablas no surgen efecto hasta que se reinicia el sistema.

4.1. Driver de comunicaciones

Para dar soporte a las comunicaciones del servidor con los ARP se utilizó un driver de comunicaciones genérico sobre protocolos TCP/UDP ya desarrollado para aplicaciones previas, realizando pequeñas modificaciones para adaptarlo a las peculiaridades de EOCEN.

Este driver se diseñó como una librería dinámica para funcionar sobre plataformas Linux y se implementó en C++.

En el presente apartado se describen los mecanismos de funcionamiento del driver, el uso que realiza de los distintos protocolos de comunicaciones, los procesos de extracción de datos de las tramas y de generación de nuevas tramas y los parámetros de configuración del driver.

4.1.1. Uso de los protocolos

Las comunicaciones entre los analizadores de red del sistema y el módulo de control se realizan a través de los protocolos TCP y UDP.

Para poder establecer los enlaces entre las máquinas, es necesario realizar previamente la configuración de la red, asignando a cada máquina una dirección IP, que debe ser estática. Para ello se hizo uso de un servidor DHCP [9] instalado en el equipo principal EOCEN.

Este servidor se configuró para establecer las direcciones IP de cada máquina en función de la MAC de ésta, editando el fichero /etc/*dhcpd.conf*, al que se le añadieron las siguientes líneas correspondientes a los equipos del prototipo experimental del sistema:

```
host EOCEN {
      hardware ethernet 08:00:2b:4c:59:23;
      fixed-address 192.168.104.25;
}
host ARP002 {
      hardware ethernet 08:00:2b:4c:52:78;
      fixed-address 192.168. 104.26;
}
host ADS002 {
      hardware ethernet 08:00:2b:4c:d0:3b;
      fixed-address 192.168. 104.27;
}
host ARP412 {
      hardware ethernet 08:00:2b:4c:5b:a7;
      fixed-address 192.168. 104.28;
}
host ADS412 {
      hardware ethernet 08:00:2b:d1:0d:30;
      fixed-address 192.168. 104.29;
}
```

Los ARP fueron programados para, tras un reset o tras una “perdida de link” en el cable, solicitar su dirección al servidor DHCP. Además, por motivos de mantenimiento del enlace, se

decidió actualizar la dirección del analizador cada 5 horas, meta que se consigue a través del envió por parte del driver un mensaje de reseteo del módulo de comunicaciones de los analizadores de red a través de UDP.

Una vez determinada la configuración de red, se establecen, a petición del equipo cliente, conexiones TCP y UDP para el intercambio de mensajes de información y configuración:

- **Comunicación por UDP**: los mensajes UDP fueron desarrollados para sondear la red en busca de nuevos equipos conectados y para verificar si es posible alcanzar dichos equipos desde el servidor. También se pueden utilizar para testear el estado de la comunicación con alguno de los ARP ya dados de alta en la red.

  El servidor principal de EOCEN es el encargado de llevar la iniciativa de la comunicación lanzando un mensaje UDP cada vez que estime oportuno. Este mensaje puede distribuirse por broadcast, llegando a todos los ARP's instalados en la misma red, o bien ser dirigido a un equipo concreto para determinar si está accesible.

  El ARP, al recibir una trama UDP envía un asentimiento, del que el servidor obtendrá información de sus direcciones físicas y direcciones IP para establecer posteriormente las conexiones TCP.

  Cabe reseñar que los puertos utilizados para la comunicación por UDP, tanto en el servidor principal como en los ARP, son configurables por el usuario.

  La estructura de los mensajes UDP es la siguiente:

Campo	Tamaño
Cabecera identificativa: "GPTarp"	6 Bytes

Tabla 3: Estructura de la trama UDP de sondeo de red

Campo	Tamaño
Cabecera identificativa: "gptARP"	6 Bytes
Dirección MAC del analizador en hexadecimal	6 Bytes
Dirección IP del analizador en hexadecimal	4 Bytes

Tabla 4: Estructura de la trama de asentimiento UDP

El otro tipo de mensaje UDP contemplado en la comunicación entre el servidor y los ARP es el destinado a reinicializar el analizador. La utilidad principal de este proceso es el que el ARP pida mediante DHCP una IP nueva, después de algún cambio en la configuración del sistema.

La estructura de estos mensajes es:

Campo	Tamaño
Cabecera identificativa: “ReSeT”	6 Bytes

Tabla 5: Estructura de la trama UDP de reseteo de ARP

- **Comunicación por TCP**: los analizadores de red poseen un puerto TCP (dirección configurable) preparado para recibir conexiones entrantes. El servidor EOCEN es el que toma la iniciativa de crear dicha conexión, actuando como cliente TCP. Una vez establecido el canal de comunicación, el analizador mandará tramas de datos al sistema sin previa petición por parte del driver y sin mensaje de asentimiento.

  Este proceso se simultanea con el envío de mensajes de peticiones y de configuración del analizador a iniciativa del driver, que serán asentidos por el ARP. Todos estos mensajes, además de las transmisiones de cyclics y de estudios de calidad por parte del ARP, se programaron para tener prioridad sobre las tramas de datos, de manera que las tramas de información que lleguen mientras se está recibiendo uno de estos mensajes se descartan.

4.1.2. Extracción de los datos de las tramas recibidas

Las tramas enviadas por los analizadores instalados en el sistema al driver de comunicaciones poseen distinto tamaño y composición según la información a transmitir. Por ello, para poder extraer los datos que portan es necesario conocer su estructura y composición.

El proceso de obtención de los datos de las tramas queda recogido en el diagrama de flujo de la imagen 46:

Al recibir una trama, el driver discrimina el tipo de la misma mediante su cabecera identificativa, y establece el tamaño y posición de los parámetros que contiene. Tras comprobar que la trama ha llegado correctamente (tamaño esperado), obtiene de manera secuencial los valores de los distintos campos, asociando a cada uno, según su posición, los

correspondientes identificadores de puerta. Cada vez que extrae un dato, lo envía al núcleo que se encarga de, en función de la tabla interna de variables subscritas por cada plugin, distribuirla al resto de módulos del sistema. Finalmente, el driver borra la trama y se pone a la espera de la siguiente.

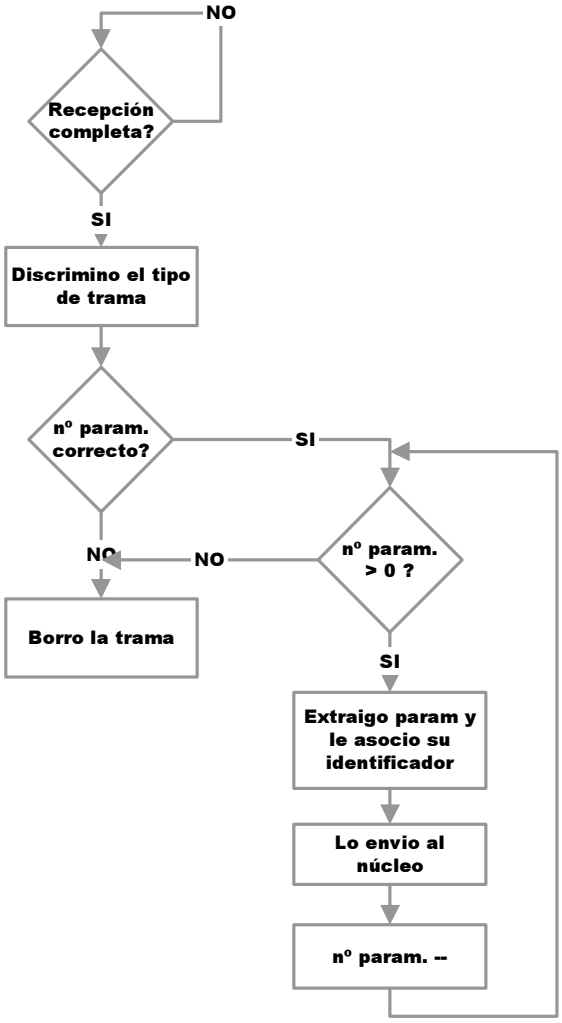


Imagen 46: Diagrama de flujo del proceso de recuperación de los datos de las tramas

Las posibles tipos de tramas que puede recibir el driver de los ARP del sistema son:

- **Tramas de datos**: Las tramas de datos enviadas por los ARP contienen la información necesaria para que desde el interfaz visual del sistema se pueda monitorizar los parámetros eléctricos trifásicos y de los canales controlados por los ADS de todas las máquinas instaladas.

  A la hora de descomponer la trama para obtener la información que porta, hay que tener en cuenta que el tamaño y número de parámetros de la misma es configurable,

pudiendo portar sólo los datos recogidos por el ARP o añadir a los anteriores los controlados por el/los ADS asociados a cada ARP. En el primer caso, la longitud de la trama de datos es de 2403 Bytes, mientras que en el segundo alcanza los 2931 Bytes.

Hay que reseñar que el microprocesador en el que se basa la comunicación del ARP limita la longitud de las tramas a enviar a 1452 Bytes, por lo que se necesitan varias tramas para enviar toda la información. Esto no implica ningún problema, ya que el protocolo TCP asegura la recepción en orden correcto de los paquetes. De esta manera, los métodos de recepción de los datos no tienen más que esperar el número de Bytes asociado a la cabecera de la trama, sin necesidad de trocear y numerar las tramas.

La estructura de las tramas de datos, en el caso de sólo contener datos del ARP es la siguiente:

Campo	Tamaño
Cabecera identificativa: 0,0,0	3 Bytes
Onda Vr en enteros de 16 bits (ajustado valor absoluto máximo a 32768)	256 Bytes
Onda Vs (idem)	256 Bytes
Onda Vt (ídem)	256 Bytes
Onda Iu (idem)	256 Bytes
Onda Iv (idem)	256 Bytes
Onda Iw (ídem)	256 Bytes
Espectro de Vr en enteros sin signo de 16 bits (expresado en tanto por 10000 respecto al rms verdadero)	100 Bytes
Espectro de Vs (ídem)	100 Bytes
Espectro de Vt (ídem)	100 Bytes
Espectro de Iu (ídem)	100 Bytes
Espectro de Iv (ídem)	100 Bytes
Espectro de Iw (ídem)	100 Bytes
Tensión Vr rms (flotante)	4 Bytes
Tensión Vs rms (flotante)	4 Bytes
Tensión Vt rms (flotante)	4 Bytes
Corriente Iu rms (flotante)	4 Bytes
Corriente Iv rms (flotante)	4 Bytes
Corriente Iw rms (flotante)	4 Bytes
Tensión Vrs rms (flotante)	4 Bytes
Tensión Vst rms (flotante)	4 Bytes
Tensión Vtr rms (flotante)	4 Bytes
Potencia activa generada fase R (flotante)	4 Bytes
Potencia activa generada fase S (flotante)	4 Bytes
Potencia activa generada fase T (flotante)	4 Bytes
Potencia activa consumida fase R (flotante)	4 Bytes
Potencia activa consumida fase S (flotante)	4 Bytes
Potencia activa consumida fase T (flotante)	4 Bytes
Potencia reactiva inductiva fase R (flotante)	4 Bytes
Potencia reactiva inductiva fase S (flotante)	4 Bytes
Potencia reactiva inductiva fase T (flotante)	4 Bytes
Potencia reactiva capacitiva fase R (flotante)	4 Bytes
Potencia reactiva capacitiva fase S (flotante)	4 Bytes

Potencia reactiva capacitiva fase T (flotante)	4 Bytes
Potencia aparente fase R (flotante)	4 Bytes
Potencia aparente fase S (flotante)	4 Bytes
Potencia aparente fase T (flotante)	4 Bytes
Factor de potencia fase R (flotante)	4 Bytes
Factor de potencia fase S (flotante)	4 Bytes
Factor de potencia fase T (flotante)	4 Bytes
THD Vr (flotante)	4 Bytes
THD Vs (flotante)	4 Bytes
THD Vt (flotante)	4 Bytes
THD Iu (flotante)	4 Bytes
THD Iv (flotante)	4 Bytes
THD Iw (flotante)	4 Bytes
Tensión fase-neutro rms trifásica (flotante)	4 Bytes
Corriente rms trifásica (flotante)	4 Bytes
Tensión fase-fase rms trifásica (flotante)	4 Bytes
Potencia activa generada trifásica (flotante)	4 Bytes
Potencia activa consumida trifásica (flotante)	4 Bytes
Potencia reactiva inductiva trifásica (flotante)	4 Bytes
Potencia reactiva capacitiva trifásica (flotante)	4 Bytes
Potencia aparente trifásica (flotante)	4 Bytes
Factor de potencia trifásico (flotante)	4 Bytes
Energía activa generada fase R (flotante)	4 Bytes
Energía activa generada fase S (flotante)	4 Bytes
Energía activa generada fase T (flotante)	4 Bytes
Energía activa consumida fase R (flotante)	4 Bytes
Energía activa consumida fase S (flotante)	4 Bytes
Energía activa consumida fase T (flotante)	4 Bytes
Energía reactiva inductiva fase R (flotante)	4 Bytes
Energía reactiva inductiva fase S (flotante)	4 Bytes
Energía reactiva inductiva fase T (flotante)	4 Bytes
Energía reactiva capacitiva fase R (flotante)	4 Bytes
Energía reactiva capacitiva fase S (flotante)	4 Bytes
Energía reactiva capacitiva fase T (flotante)	4 Bytes
Energía activa generada trifásica (flotante)	4 Bytes
Energía activa consumida trifásica (flotante)	4 Bytes
Energía reactiva inductiva trifásica (flotante)	4 Bytes
Energía reactiva capacitiva trifásica (flotante)	4 Bytes
Frecuencia (flotante)	4 Bytes
Factor de escala de onda Vr (flotante)	4 Bytes
Factor de escala de onda Vs (flotante)	4 Bytes
Factor de escala de onda Vt (flotante)	4 Bytes
Factor de escala de onda Iu (flotante)	4 Bytes
Factor de escala de onda Iv (flotante)	4 Bytes
Factor de escala de onda Iw (flotante)	4 Bytes
Fecha en entero sin signo de 32 bits (segundos desde 00:00 de 1/1/1980)	4 Bytes
Cabecera identificativa: ‘F’,’D’,’T’	3 Bytes

Tabla 6: Campos que contiene la trama de información del ARP. Se muestran en el mismo orden en que aparecen en la trama

Si además incorpora los datos del ADS, antes de la cabecera identificativa de final de trama se añaden los siguientes campos:

Campo	Tamaño
Espectro de señal canal 1 en ADS en enteros sin signo de 16 bits (expresado en tanto por 10000 respecto al rms verdadero)	128 Bytes
Espectro de señal canal 2 en ADS (ídem)	128 Bytes
Espectro de señal canal 3 en ADS (ídem)	128 Bytes
Espectro de señal canal 4 en ADS (ídem)	128 Bytes
Valor rms de señal canal 1 en ADS (flotante)	4 Bytes
Valor rms de señal canal 2 en ADS (flotante)	4 Bytes
Valor rms de señal canal 3 en ADS (flotante)	4 Bytes
Valor rms de señal canal 4 en ADS (flotante)	4 Bytes
Cabecera identificativa: 'F','D','T'	3 Bytes

Tabla 7: Campos que se añaden a la trama de información del ARP cuando esta porta los datos del ADS

- **Trama de alarmas**: Cada vez que se produce un evento programable se origina el envío de un cyclic hacia el sistema EOCEN sin previa petición por parte del servidor EOCEN. También se recoge la posibilidad del envío de cyclics como respuesta a una petición por parte del operador. En este caso, el identificador del parámetro eléctrico será 0.

  El cyclic contiene los datos almacenados en la memoria del ARP sobre las tres tensiones e intensidades trifásicas, que consisten en 128 muestras (Bytes) de cada forma de onda tomadas durante un segundo.

  Para poder enviar tal cantidad de información, se decidió utilizar una estructura de multitrama formada por tramas de 1020 Bytes de información efectivos (sin contar con las cabeceras ni la información de supertrama). Esto supone la adición de nuevos campos necesarios para soportar la estructura de la supertrama: numero de tramas que la componen, numeración secuencial de las tramas para permitir reconstruir la multitrama en el receptor y la longitud de los datos en cada una, que en el caso de la última puede ser inferior a los 1020 Bytes.

  La estructura de las tramas queda recogida en la imagen número 47.

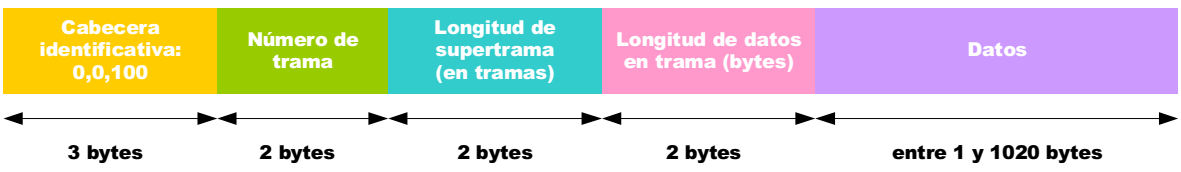


Imagen 47: Estructura de las tramas de alarmas

La información incluida en el campo "*Datos*" se organiza de forma inversa a como se recogió, es decir, primero se transmite la muestra de la forma de onda correspondiente al instante temporal más alejado. Todas las muestras se envían escaladas por factores que se proporcionan en la misma trama.

Además de estos datos, se incluye el identificador del parámetro eléctrico que provoca el evento, el valor del mismo y la fecha y hora en que se produce.

Campos contenidos en Datos	Tamaño
Fecha en entero sin signo de 32 bits (segundos desde 00:00 de 1/1/1980)	4 Bytes
Índice del parámetro eléctrico que provoca el evento (entero de 4 Bytes). El índice se establece según la tabla del Anexo C	4 Bytes
Valor tomado por el parámetro que provoca el evento (flotante)	4 Bytes
Escalado de las muestras de Vr (flotante)	4 Bytes
Escalado de las muestras de Vs (flotante)	4 Bytes
Escalado de las muestras de Vt (flotante)	4 Bytes
Escalado de las muestras de Iu (flotante)	4 Bytes
Escalado de las muestras de Iv (flotante)	4 Bytes
Escalado de las muestras de Iw (flotante)	4 Bytes
Muestra Onda Vr (instante temporal n)	2 Bytes
Muestra Onda Vs (instante temporal n)	2 Bytes
Muestra Onda Vs (instante temporal n)	2 Bytes
Muestra Onda Iu (instante temporal n)	2 Bytes
Muestra Onda Iv (instante temporal n)	2 Bytes
Muestra Onda Iw (instante temporal n)	2 Bytes
Muestra Onda Vr (instante temporal n-1)	2 Bytes
Muestra Onda Vs (instante temporal n-1)	2 Bytes
Muestra Onda Vs (instante temporal n-1)	2 Bytes
Muestra Onda Iu (instante temporal n-1)	2 Bytes
Muestra Onda Iv (instante temporal n-1)	2 Bytes
Muestra Onda Iw (instante temporal n-1)	2 Bytes
….	….
Muestra Onda Vr (instante temporal 1)	2 Bytes
Muestra Onda Vs (instante temporal 1)	2 Bytes
Muestra Onda Vs (instante temporal 1)	2 Bytes
Muestra Onda Iu (instante temporal 1)	2 Bytes
Muestra Onda Iv (instante temporal 1)	2 Bytes
Muestra Onda Iw (instante temporal 1)	2 Bytes

Tabla 8: Datos contenidos en el campo "Datos" de la trama de alarmas

- **Trama de estudio de calidad**: Un estudio de calidad está formado por las medidas, en tanto por diez mil, de las líneas espectrales resultantes de realizar FFT´s a 10 ventanas temporales de 16 ciclos de red, según las normas establecidas para los estudios de 3 segundos en el RD 436/2004, a las tres magnitudes de tensión. De esta manera, se dispone información de 50 armónicos de cada tensión y para cada uno de ellos, de los 15 ínter armónicos asociados. En total, la información a transmitir es de 9600 Bytes.

Debido a la enorme cantidad de información a transmitir, se decidió nuevamente utilizar una estructura multitrama formada por doce tramas con 800 Bytes de información útiles en cada una.

La estructura de la multitrama queda reflejada en la imagen 48:

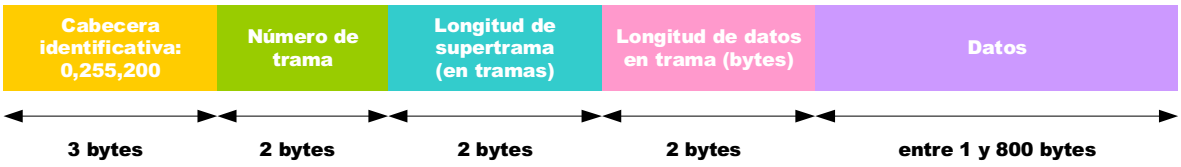


Imagen 48: Estructura de las tramas para el envío de estudios de calidad

Los datos se organizan de manera secuencial por tensiones, enviándose primero el conjunto de datos de Vr, a continuación el de Vs y finalmente el de Vt. Dentro de cada conjunto, la secuencia de los datos es la siguiente:

Orden de la información del grupo	Tamaño
Ínter armónico 0_1 Vx	4 Bytes
Ínter armónico 0_2 Vx	4 Bytes
Ínter armónico 0_3 Vx	4 Bytes
…	…
Ínter armónico 0_{15} Vx	4 Bytes
Armónico 1 Vx	4 Bytes
Ínter armónico 1_1 Vx	4 Bytes
Ínter armónico 1_2Vx	4 Bytes
…	…
Ínter armónico 1_{15} Vx	4 Bytes
Armónico 2 Vr	4 Bytes
…	…
Armónico 50 Vr (flotante)	4 Bytes

Tabla 9: Orden de los datos en cada grupo

4.1.3. Generación de las tramas a enviar por el driver

Para dotar al sistema de bidireccionalidad en la comunicación y permitir así el envió de órdenes necesario para realizar las tareas de control, se requiere que el driver de comunicaciones sea capaz de generar y enviar tramas de información.

El proceso de envío de una trama comienza estableciendo una comunicación por TCP con el ARP deseado. Una vez establecido el enlace, el driver de comunicaciones genera la trama

necesaria en función del tipo de información a enviar, abre un s*tream* hacia el ARP y envía los datos de forma secuencial.

Al igual que ocurre en la extracción de datos de las tramas recibidas, para que el ARP destinatario de la trama sea capaz de interpretar la información que se le envía, es preciso establecer un formato común en el que la cantidad y orden de los parámetros esté determinado. Los formatos de trama definidos son:

- **Tramas de configuración**: Los mensajes de configuración permiten cambiar determinados parámetros que rigen el funcionamiento de los ARP de forma remota.

  A diferencia del resto de los mensajes, debido a que su contenido es crítico, se estableció un mecanismo de asentimiento por parte del ARP que garantice el trasvase correcto de la información.

  La estructura de estos mensajes consta de una cabecera identificativa seguida de los parámetros específicos para establecer la nueva configuración. Los tipos de mensajes de comunicación definidos son:

  - Configuración del origen de datos: permite establecer si las tramas de datos enviadas por el ARP incluyen la información generada por el ADS o no.

Campo	Tamaño
Cabecera identificativa: 255,0,200	3 Bytes
Origen de datos: 0 si sólo ARP, 1 si ARP+ADS	1 Byte

Tabla 10: Estructura de la trama de configuración del origen de datos

Campo	Tamaño
Cabecera identificativa: 255,0,200	3 Bytes
Cabecera identificativa: ‘F’,’D’,’T’	3 Bytes

Tabla 11: Estructura de la trama de asentimiento a la configuración del origen de datos

  - Configuración de las relaciones de transformación: permite establecer las relaciones de transformación para las tensiones e intensidades.

Campo	Tamaño
Cabecera identificativa: 255,0,0	3 Bytes

Numerador de relación de transformación de tensión (entero 32 bits)	4 Bytes
Denominador de relación de transformación de tensión (entero 32 bits)	4 Bytes
Numerador de relación de transformación de corriente (entero 32 bits)	4 Bytes
Denominador de relación de transformación de corriente (entero 32 bits)	4 Bytes

Tabla 12: Estructura de la trama de configuración de las relaciones de transformación

Campo	Tamaño
Cabecera identificativa: 255,0,0	3 Bytes
Cabecera identificativa: ‘F’,’D’,’T’	3 Bytes

Tabla 13: Estructura de la trama de asentimiento a la configuración de las relaciones de transformación

- Mensaje de configuración de eventos programables: permite establecer un disparador para la generación de eventos programables.

Campo	Tamaño
Cabecera identificativa: 255,0,100	3 Bytes
Número de evento a programar (1,2 ó 3)	1 Byte
Evento X: 1 si alarma en Domingo, 0 si no	1 Byte
Evento X: 1 si alarma en Lunes, 0 si no	1 Byte
Evento X: 1 si alarma en Martes, 0 si no	1 Byte
Evento X: 1 si alarma en Miércoles, 0 si no	1 Byte
Evento X: 1 si alarma en Jueves, 0 si no	1 Byte
Evento X: 1 si alarma en Viernes, 0 si no	1 Byte
Evento X: 1 si alarma en Sábado, 0 si no	1 Byte
Evento X: Inicio de franja horaria diaria (segundos desde las doce de la noche) en entero sin signo de 32 bits	4 Bytes
Evento X: fin de franja horaria diaria (segundos desde las doce de la noche) en entero sin signo de 32 bits	4 Bytes
Evento X: sentido de la comparación para determinar evento. 0 si menor igual. 1 si mayor	1 Byte
Evento X: cota del parámetro eléctrico que determina si hay o no evento (flotante)	4 Bytes
Evento X: índice del parámetro eléctrico que provoca el evento (ver Anexo C)	1 Byte
Evento X:1 habilita el evento. 0 lo deshabilita	1 Byte

Tabla 14: Estructura de la trama configuración de eventos programables

Campo	Tamaño
Cabecera identificativa: 255,0,100	3 Bytes
Número de evento a programar (1,2 ó 3). 0 si no se	1 Byte

modifica ningún evento. Cabecera identificativa: ‘F’,’D’,’T’	3 Bytes

Tabla 15: Estructura de la trama de asentimiento a la configuración de eventos programables

- **Tramas de petición:** se utilizan para solicitar al ARP que ejecute una acción determinada, que puede ser:

  - Petición de puesta a cero de los contadores de energía del ARP. En este caso, se requiere de una trama de asentimiento.

Campo	Tamaño
Cabecera identificativa: 0,255,0	3 Bytes

Tabla 16: Estructura de la trama de petición de puesta a cero de los contadores de energía

Campo	Tamaño
Cabecera identificativa: 255,0,100	3 Bytes
Cabecera identificativa: ‘F’,’D’,’T’	3 Bytes

Tabla 17: Estructura de la trama de asentimiento a la petición de puesta a cero de los contadores de energía

  - Petición de envío de un cyclic: como ya se ha comentado anteriormente, el sistema permite el envío de cyclics bajo petición. La estructura del cyclic enviado es la misma que cuando se produce por un evento programable (ver imagen 47 y tabla 8). Para distinguir la causa del cyclic usaremos cabeceras distintas para cada tipo: 0,0,100 para los producidos por un evento y 0,255,100 para los enviados bajo petición.

Campo	Tamaño
Cabecera identificativa: 0,255,100	3 Bytes

Tabla 18: Estructura de la trama de petición de cyclic

  - Petición de estudio de calidad: los estudios de calidad se envían al sistema bajo demanda. El driver de comunicaciones está programado para solicitar el envío de los estudios con una periodicidad configurable a través del parámetro

DriverEOCEN_periodo_est_calidad de la tabla eocen_conf.conf_parametros de la base de datos, que por defecto está fijado en 540 segundos.

Campo	Tamaño
Cabecera identificativa: 0,255,200	3 Bytes

Tabla 19 Estructura de la trama de petición de un estudio de calidad

Debido a la gran cantidad de información de la que consta un estudio de calidad, la transmisión simultánea de varios de ellos puede llegar a saturar el sistema, ya que mientras que se está recibiendo uno se descartan todas las tramas de datos que llegan al driver. Para evitar estas saturaciones, se definió una nueva orden que permite solicitar al ARP la parada de la transmisión de un estudio de calidad:

Campo	Tamaño
Cabecera identificativa: 0,255,50	3 Bytes

Tabla 20: Estructura de la trama de petición de parada de un estudio de calidad

- Petición de sincronización horaria del ARP: permite establecer la fecha y hora del reloj interno del ARP:

Campo	Tamaño
Cabecera identificativa: 0,255,1	3 Bytes
Fecha y hora actual: (número de segundos desde 01/01/1980 a las 0:00:00)	4 Bytes

Tabla 21: Estructura de la trama de petición de sincronización

- Petición de consulta de eventos programables: permite recuperar la configuración de un determinado evento programado en el ARP consultado:

Campo	Tamaño
Cabecera identificativa: 0,255,10	3 Bytes
Numero del evento programable (1, 2 ó 3)	3 Bytes

Tabla 22: Estructura de la trama de petición de consulta de la configuración de un evento programable

Como respuesta, el ARP envía un mensaje igual al presentado en la tabla 14 pero en el que la cabecera identificativa es 0,255,10.

- **Trama de control**: existe un mecanismo de seguridad, para la desconexión automática en caso de no detectarse comunicación en el enlace TCP. Se trata de un sistema temporizador, denominado *Keep Alive Timer,* y que consiste en el envío por parte del driver de comunicaciones de una trama de control cada minuto. El ARP se encarga de monitorizar la llegada de dichas tramas, abortando la comunicación con el sistema si se supera el tiempo máximo de espera entre dos tramas *Keep Alive Timer,* fijado en 5 minutos. La estructura de la trama de control queda recogida en la siguiente tabla:

Campo	Tamaño
Cabecera identificativa: 0,255,255	3 Bytes

Tabla 23: Estructura de la trama de control

El control de la comunicación en el extremo del driver se realiza contabilizando el tiempo que transcurre entre la llegada de dos tramas de datos consecutivas del mismo ARP. Caso de superarse los 5 minutos, el driver finaliza automáticamente la conexión con dicho ARP, intentando reestablecerla de nuevo cada 3 minutos.

4.1.4. Configuración del driver

El driver de comunicaciones requiere de una serie de parámetros para configurar su funcionamiento. Estos se obtienen de las tablas de *eocen_conf* del a base de datos al iniciarse el sistema. Y pueden ser modificados por el administrador del sistema.

Los parámetros configurables son:

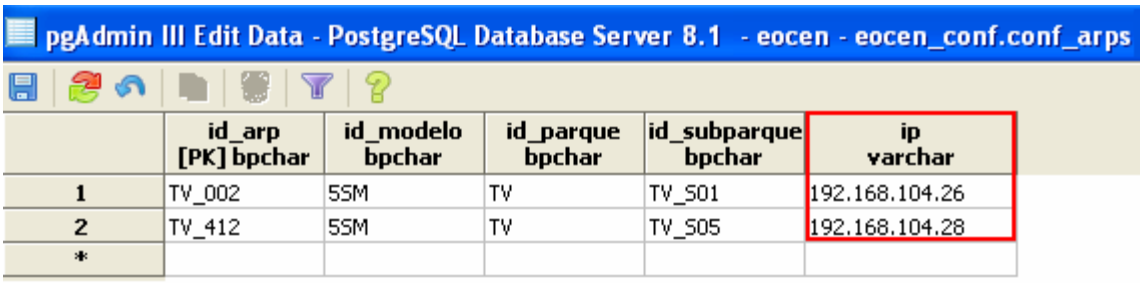
- **Direcciones IP asignadas a cada analizador de red**: como ya se ha comentado anteriormente, las direcciones IP de cada máquina del sistema son asignadas estáticamente por un servidor DHCP.

  El driver de comunicaciones necesita conocer la dirección de los analizadores de red para poder establecer las conexiones TCP con ellos. Como no puede acceder al servidor DHCP para consultarlas, requiere de la existencia de una copia de la tabla de direcciones asignadas en la base de datos. Las IP de los ARP se encuentran en la tabla *conf_arp* mientras que para los ADS, se estableció como convenio que llevarían como dirección IP la siguiente a la asignada al ARP al que están asociados.

Para el sistema prototipo, la configuración que se estableció es la siguiente:

Máquina	IP
Servidor EOCEN	192.168.104.25
ARP aerogenerador 002	192.168.104.26
ADS aerogenerador 002	192.168.104.27
ARP aerogenerador 412	192.168.104.28
ADS aerogenerador 412	192.168.104.29

Tabla 24: Configuración de red de los elementos del sistema prototipo



	id_arp [PK] bpchar	id_modelo bpchar	id_parque bpchar	id_subparque bpchar	ip varchar
1	TV_002	5SM	TV	TV_S01	192.168.104.26
2	TV_412	5SM	TV	TV_S05	192.168.104.28
*					

Imagen 49: Direcciones de red de los ARP contenidas en *eocen_conf.conf_arps*

- **Puertos para las comunicaciones:** para poder establecer las conexiones TCP y UDP es necesario establecer por qué puertos se realizarán. La configuración de estos parámetros se encuentra en la tabla *eocen_conf.conf_parametros.*

  Para el sistema prototipo se establecieron los siguientes puertos:

Comunicación	Puerto
Conexión TCP	2712
Envío UDP	4440
Escucha UDP	4440

Tabla 25: Configuración de los puertos empleados en las comunicaciones

- **IP y periodicidad del broadcast**: la función de broadcast se emplea para descubrir posibles nuevos analizadores de red instalados en el sistema o reconectarse a aquellos con los que se ha perdido la comunicación. Para ello, se envían periódicamente tramas a la dirección de difusión y se espera la respuesta por parte de los ARP.

La configuración del periodo y la IP del broadcast se encuentran en la tabla *eocen_conf.conf_parametros.* Para el sistema prototipo se establecieron los siguientes parámetros.

Parámetro	Valor
IP difusión	192.168.255.255
Periodo	5 minutos

Tabla 26: Configuración de los parámetros de broadcast del sistema prototipo

- **Periodicidad de solicitud de datos a los ARP:** los analizadores de red transmiten los estudios de calidad bajo demanda, por lo que es preciso enviarles periódicamente peticiones de estudios.

  Los periodos de petición de datos se encuentran configurados en la tabla *eocen_conf.conf_parametros.* Para el prototipo experimental se establecieron los siguientes valores:

Solicitud	Periodo
Estudio de calidad	540 segundos

Tabla 27: Configuración de los parámetros de broadcast del sistema prototipo.

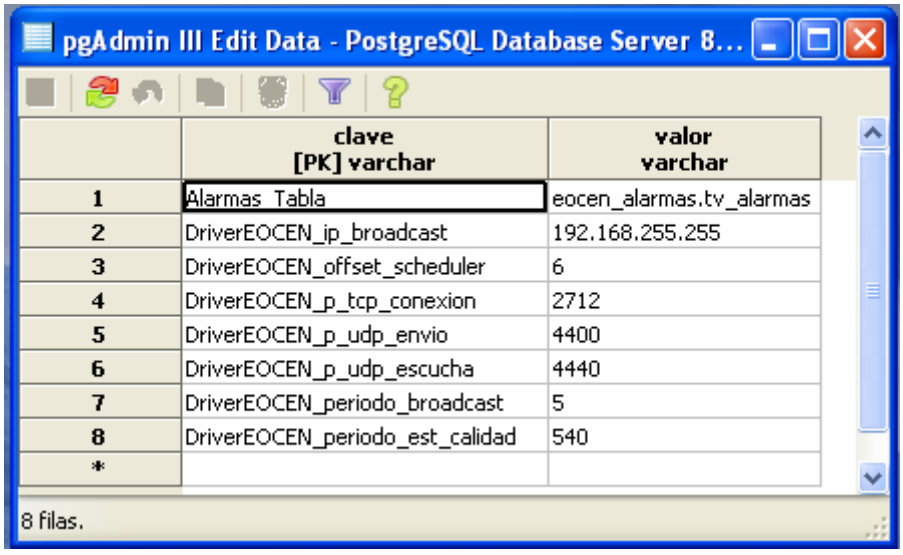


Imagen 50: Contenido de la tabla *conf_parametros*

4.2. Módulo de bombeo

El módulo de bombeo es el plugin encargado de insertar periódicamente las variables recibidas por el driver de comunicaciones o resultantes de las operaciones llevadas a cabo por el módulo de cálculo en la base de datos.

El tipo de bombeo realizado por este módulo es multivariable por tabla, es decir, en una tabla se almacenan los valores de múltiples variables, provenientes de un único origen de datos, en diferentes instantes del tiempo.

La clave primaria de las tablas es la fecha de los datos, que se representa mediante el llamado *formato autónomo***,** utilizando una representación completa de la fecha, la hora, y el huso horario en un único campo *fecha_hora*. Cada una de las filas puede contener más de un dato compartiendo la misma marca de tiempo (tabla agrupada), identificándose la variable por el nombre de la columna.

Al igual que el resto de módulos del núcleo fue desarrollado como una librería dinámica en ANSI C++.

En este apartado se detalla el funcionamiento del módulo así como los posibles parámetros configurables del mismo.

4.2.1. Configuración del módulo de bombeo

Para poder funcionar, el módulo de bombeo necesita una serie de parámetros que le proporcionen la información necesaria sobre las variables a las que tiene que subscribirse en el núcleo, las tablas en las que debe almacenar cada variable y la periodicidad de las inserciones de cada una.

Los campos a configurar son:

- **id_puerta**: código numérico con el que el módulo de bombeo identifica unívocamente a cada variable a bombear.
- **tabla**: nombre completo, incluido el esquema, de la tabla donde se escribirán los valores de la variable.
- **agrupada**: indica si la tabla es agrupada (1) o no (0).

- **columna**: especifica para las tablas agrupadas, el nombre de la columna correspondiente a la variable.

- **periodicidad**: intervalo de tiempo entre dos inserciones consecutivas de la variable. Si se especifica periodicidad 0, las inserciones se realizan nada más recibir el dato.

  Por convenio, se decidió configurar el bombeo de todas las variables cada 5 segundos, salvo en el caso de las referentes a los estudios de calidad. Debido a la gran cantidad de información que contienen los estudios de calidad, para evitar saturar la base de datos, se decidió ralentizar las inserciones de las variables relacionadas con los estudios, estableciendo para ellas una periodicidad de 10 minutos.

- **escala**: valor por el que se multiplicará la variable antes de ser escrita en la base de datos.

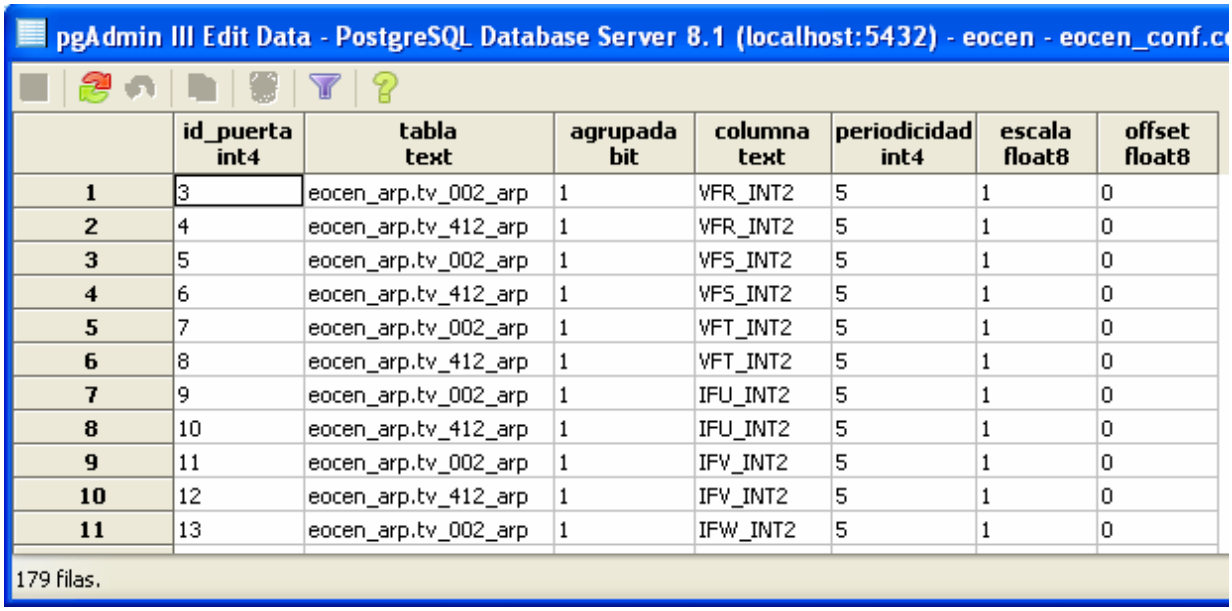
- **offset**: valor que se sumará a la variable después de ser multiplicada por la escala y antes de ser escrita en la base de datos.

La carga de todos estos valores se realiza al inicializarse el módulo desde las tablas y vistas de *eocen_conf*:

- **conf_bombeo_inm_view**: contiene los parámetros de bombeo de las variables de ARP, ADS, alarmas y resúmenes de parque. Los datos de esta vista se obtienen de las tablas *eocen_conf.conf_puertas* y *eocen_conf.conf_bombeo_inm* mediante el siguiente script:

```
CREATE OR REPLACE VIEW eocen_conf.conf_bombeo_inm_view AS
 SELECT cp.id_puerta, (((pb.esquema::text || '.'::text) || pb.prefijo_tabla::text) ||
substr(cp.nombre::text, 12, 3)) || pb.sufijo_tabla::text AS tabla, pb.agrupada,
substr(cp.nombre::text, 23, 8) AS columna, pb.periodicidad, pb.escala, pb."offset"
   FROM eocen_conf.conf_puertas cp, eocen_conf.conf_bombeo_inm pb
  WHERE cp.nombre ~~ pb.patron_puerta::text;

ALTER TABLE eocen_conf.conf_bombeo_inm_view OWNER TO eocen;
```



pgAdmin III Edit Data - PostgreSQL Database Server 8.1 (localhost:5432) - eocen - eocen_conf.c

	id_puerta int4	tabla text	agrupada bit	columna text	periodicidad int4	escala float8	offset float8
1	3	eocen_arp.tv_002_arp	1	VFR_INT2	5	1	0
2	4	eocen_arp.tv_412_arp	1	VFR_INT2	5	1	0
3	5	eocen_arp.tv_002_arp	1	VFS_INT2	5	1	0
4	6	eocen_arp.tv_412_arp	1	VFS_INT2	5	1	0
5	7	eocen_arp.tv_002_arp	1	VFT_INT2	5	1	0
6	8	eocen_arp.tv_412_arp	1	VFT_INT2	5	1	0
7	9	eocen_arp.tv_002_arp	1	IFU_INT2	5	1	0
8	10	eocen_arp.tv_412_arp	1	IFU_INT2	5	1	0
9	11	eocen_arp.tv_002_arp	1	IFV_INT2	5	1	0
10	12	eocen_arp.tv_412_arp	1	IFV_INT2	5	1	0
11	13	eocen_arp.tv_002_arp	1	IFW_INT2	5	1	0

179 filas.

Imagen 51: Contenido de la vista *conf_bombeo_inm_view*

- **conf_bombeo**: almacena los parámetros de bombeo de las variables de los estudios de calidad. Los campos de esta vista se obtienen de las tablas *conf_puertas*, *conf_bombeo_calidad_i* y *conf_bombeo_calidad_a* pertenecientes al esquema *eocen_conf* mediante el siguiente script:

```
CREATE OR REPLACE VIEW eocen_conf.conf_bombeo AS
( SELECT 'TV' AS id_bombeo, conf_bombeo_calidad_a_view.id_puerta,
conf_bombeo_calidad_a_view.tabla, conf_bombeo_calidad_a_view.agrupada,
conf_bombeo_calidad_a_view.columna, conf_bombeo_calidad_a_view.periodicidad,
conf_bombeo_calidad_a_view.escala, conf_bombeo_calidad_a_view."offset"
   FROM eocen_conf.conf_bombeo_calidad_a_view
UNION ALL
 SELECT 'TV' AS id_bombeo, conf_bombeo_calidad_i_view.id_puerta,
conf_bombeo_calidad_i_view.tabla, conf_bombeo_calidad_i_view.agrupada,
conf_bombeo_calidad_i_view.columna, conf_bombeo_calidad_i_view.periodicidad,
conf_bombeo_calidad_i_view.escala, conf_bombeo_calidad_i_view."offset"
   FROM eocen_conf.conf_bombeo_calidad_i_view)
UNION ALL
 SELECT 'TV' AS id_bombeo, conf_bombeo_inm_view.id_puerta,
conf_bombeo_inm_view.tabla, conf_bombeo_inm_view.agrupada,
conf_bombeo_inm_view.columna, conf_bombeo_inm_view.periodicidad,
conf_bombeo_inm_view.escala, conf_bombeo_inm_view."offset"
   FROM eocen_conf.conf_bombeo_inm_view
  ORDER BY 1;
```

pgAdmin III Edit Data - PostgreSQL Database Server 8.1 (localhost:5432) - eocen - eocen_conf.conf_bombeo

	id_bombeo text	id_puerta int4	tabla text	agrupada bit	columna text	periodicidad int4	escala float8	offset float8
1	TV	5008	eocen_calidad.tv_002_cal_10mp_escr_01	1	armonico_vR	600	1	0
2	TV	8842	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_10_vS	600	1	0
3	TV	8841	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_09_vS	600	1	0
4	TV	8840	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_08_vS	600	1	0
5	TV	8839	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_07_vS	600	1	0
6	TV	8838	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_06_vS	600	1	0
7	TV	8837	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_05_vS	600	1	0
8	TV	8836	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_04_vS	600	1	0
9	TV	8835	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_03_vS	600	1	0
10	TV	8834	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_02_vS	600	1	0
11	TV	8833	eocen_calidad.tv_412_cal_10mp_escr_41	1	intarm_01_vS	600	1	0

Imagen 52: Contenido de la tabla *conf_bombeo*

4.2.2. Funcionamiento del módulo

El funcionamiento de este plugin se basa en los métodos aportados por las clases *PuertaAgrupada*, *TablaAgrupada* y *CatRecAgrupada,* preparadas para trabajar con tablas agrupadas. El diagrama UML de las clases del plugin queda recogido en la figura 53.

Al arrancar el sistema, se produce la inicialización del módulo de bombeo. Durante este proceso, se accede a las vistas de la base de datos que contienen la configuración de las puertas y se crean los objetos tipo *PuertaAgrupada*, *TablaAgrupada* y *CategoríaRecAgrupada* necesarios para almacenar toda la información de las vistas. En el momento de su creación, los objetos tipo *CategoriaRecAgrupada* actualizan su atributo *mayor_ts_bomb* accediendo a cualquier tabla que tengan asociada y obteniendo la fecha más reciente de todas las filas (al realizarse las inserciones de forma transaccional, el valor de la fecha más reciente será igual para todas las tablas asociadas a un objeto *CategoriaRecAgrupada).*

Las referencias que se asignan entre los distintos objetos quedan recogidas en el diagrama de la figura 53.

Una vez construidos todos los objetos, se recorre la vista que define los patrones de las variables y se efectúa la suscripción de las mismas en el núcleo. A partir de ese momento, el plugin está preparado para comenzar a funcionar.

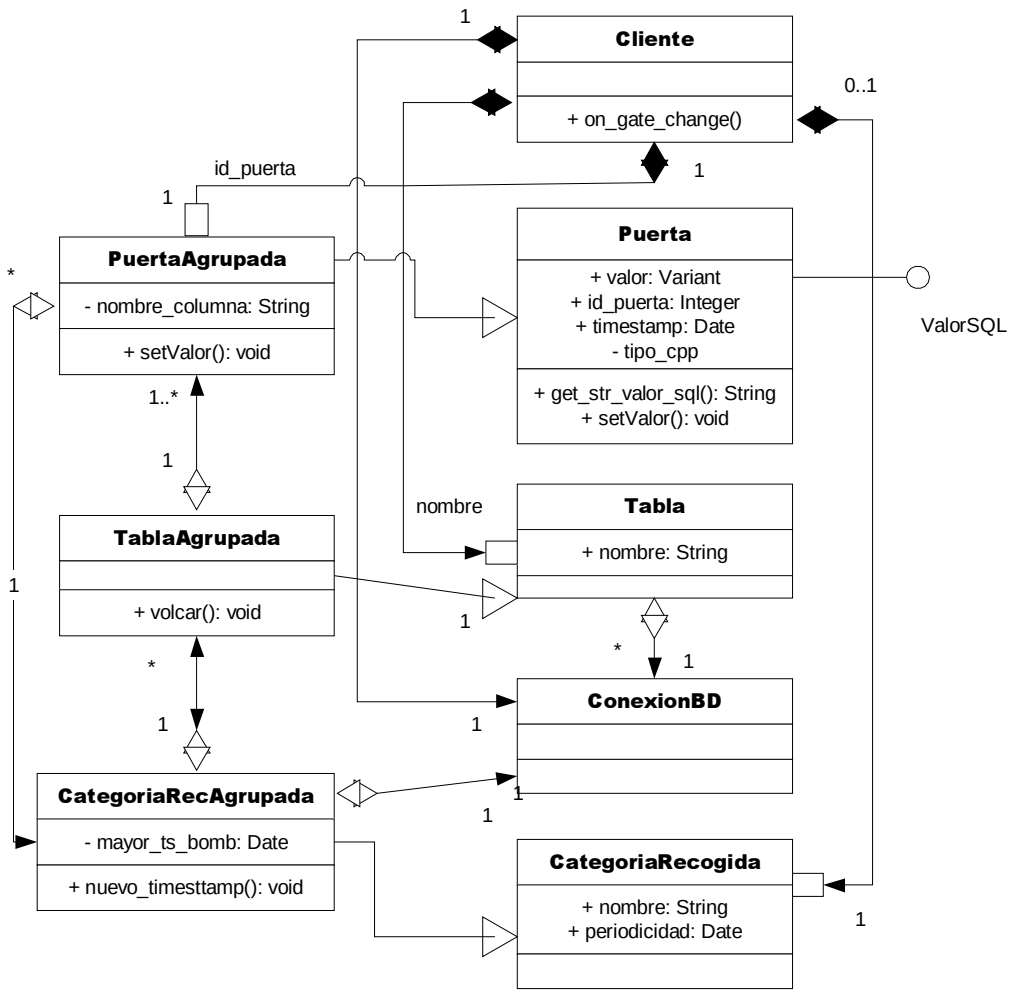


Imagen 53: Diagrama UML del módulo de bombeo

Cada vez que se produce un cambio en el valor de una puerta, debido a la llegada de un dato procedente de los analizadores de red o del módulo de cálculo, el driver busca el objeto *PuertaAgrupada* asociado a dicha variable a través de su *id_puerta* y actualiza sus atributos *valor* y *timestamp* mediante el método *set_valor().* Además, al valor recibido se le aplica el operador *typeid()* para averiguar el contenido del dato, suministrado por el núcleo en formato *variant*, y se almacena el resultado en el atributo *tipo_cpp*.

El objeto *PuertaAgrupada* notificará el nuevo *timestamp* a su *CategoríaAgrupada*. La *CategoríaAgrupada* mantiene en su variable *mayor_ts_agrupada* el mayor *timestamp* que ha sido bombeado hasta el momento. Si la suma de éste con la periodicidad de la *CategoríaAgrupada* resultase menor que el nuevo *timestamp* notificado, el plugin procede al volcado de todas las *TablasAgrupadas* asociadas a esta categoría mediante la llamada al método *volcar()*, que insertara en la base de datos tantas filas como fuera necesario para alcanzar al *timestamp* notificado.

La generación de la consulta SQL para las inserciones se hace obteniendo de los objetos *PuertaAgrupada* sus valores mediante el método *get_str_valor_sql()*. Este método convierte el atributo valor, que suministrado por el núcleo y almacenado como *variant,* en una cadena en formato SQL con la ayuda de la información del atributo *tipo_cpp.*

El volcado se efectúa dentro de una transacción abierta por la *CategoríaAgrupada*. Una vez terminada la inserción, se cierra la transacción y se actualizaría el valor de *mayor_ts_bomb*.

4.3. Módulo de cálculo

El módulo de cálculo es el plugin encargado de realizar las operaciones matemáticas necesarias sobre las variables recibidas por el núcleo antes de que se proceda a su bombeo a la base de datos.

Su principal cometido consiste en realizar los algoritmos estadísticos sobre los estudios de cálculo para obtener a partir de ellos los percentiles de los armónicos sobre los que el nuevo RD impone controles.

Al igual que el resto de plugins, es una librería dinámica implementada en ANSI C++ para funcionar sobre plataformas Linux bajo el control de una aplicación núcleo.

En este apartado se exponen los cálculos que realiza el módulo sobre las puertas así como los parámetros de configuración del mismo.

4.3.1. Cálculos que realiza el módulo

Como ya se comentó en las memorias descriptiva y justificativa del presente documento, el sistema EOCEN está implementado buscando distribuir la capacidad de procesado de la información, realizando la mayor parte del preprocesado de los datos en los analizadores de red instalados en los aerogeneradores. Esta situación reduce el volumen de operaciones a realizar por el módulo de cálculo del servidor principal, quedando únicamente encargado de calcular los factores de capacidad de los parques a partir de los datos de energías activas y reactivas, realizar los sumatorios instantáneos de energías, y de realizar los estudios estadísticos sobre los estudios de calidad recibidos.

- **Cálculos del factor de capacidad:** el factor de capacidad de un parque se define como un cociente en el que el numerador es la energía producida en un periodo del tiempo y el denominador el producto de la potencia nominal instalada en el parque por el número de horas del periodo y se expresa en tanto por ciento.

$$F(\%) = \frac{E_{prod}}{P_{nom} \times t}$$

- **Sumatorios instantáneos de energías**: Estos cálculos permiten conocer los valores globales de las energías de parque a partir de las de los aerogeneradores. Las energías calculadas son:

  - Energía reactiva actual de un aerogenerador: se calcula como la suma de las energías reactiva capacitiva y reactiva inductiva del aerogenerador.

$$E_{reac} = E_{reac\_ind} + E_{reac\_cap}$$

  - Energía reactiva total actual de parque: es la suma de las energías reactivas actuales de todos los aerogeneradores de un parque

$$E_{reac\_tota} = \sum_i E_{reac\,i}$$

  - Energía activa total generada actual: suma instantánea de todas las energías trifásicas activas generadas de los aerogeneradores que componen un parque.

$$E_{act\_tot} = \sum_i E_{act\_gen\_3f_i}$$

  - Energía activa total consumida actual: suma instantánea de todas las energías trifásicas activas consumidas de los aerogeneradores que componen un parque.

$$E_{cons\_tot} = \sum_i E_{act\_cons\_3f_i}$$

- **Estudios estadísticos de los estudios de calidad**: Cada nueve minutos el driver de comunicaciones solicita a los analizadores el envío de un estudio de calidad (resultado

del procesado de armónicos e ínter armónicos correspondiente a un estudio de 3 segundos). La trama que llega al driver dispone de la información sobre 50 armónicos y 15 ínter armónicos asociados a cada armónico de las 3 tensiones (Vr, Vs y Vt), en valores promedio, que se almacenan en la base de datos.

A partir de estos estudios de 3 segundos, el módulo de cálculo debe obtener estudios equivalentes *diezminutales* (de 10 minutos de duración). Esto se consigue realizando promedios estadísticos sobre 200 estudios de 3 segundos (200 x 3 = 600 segundos = 10 minutos).

La norma CEI 61000-3-6 impone un valor máximo para los percentiles 95% de cada uno de los armónicos de las 3 tensiones obtenidos de los estudios diezminutales (ver Anexo B).

Para obtener dichos percentiles, el módulo de cálculo va almacenando internamente los valores de cada estudio de 3 segundos en una tabla diferente. Cuando contiene 200 estudios de 3 segundos, toma los 200 valores que ocupan la posición i-ésima de cada una de las tablas, los ordena de menor a mayor y busca el que ocupa la posición 95% (200 x 0.95 = 190). Este será el percentil 95 para esa variable. El proceso se repite para todas las variables, almacenándose los percentiles en una nueva tabla de resultados a la que se le asocia la fecha y hora del último estudio de 3 segundos antes de pasar los resultados al módulo de bombeo para que los inserte en la base de datos.

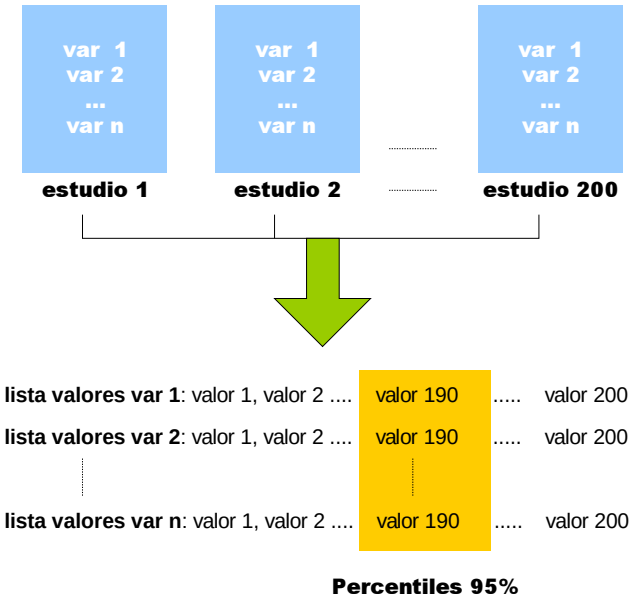


Imagen 54: Proceso de cálculo de los percentiles 95 %

4.3.2. Configuración del módulo

La configuración del módulo de cálculo se lleva a cabo a partir de tres tablas de la base de datos almacenadas en el esquema *eocen_conf.*

- *def_calculos:* tabla que contiene los tipos de cálculos implementados por el módulo: sumatorio instantáneo y cálculo de percentiles.

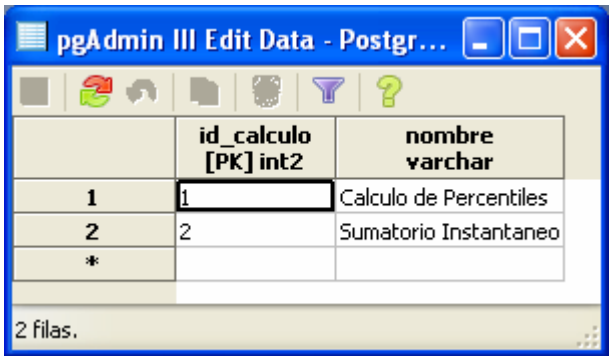


Imagen 55: Contenido de la tabla *def_calculos*

- *calc_puertascalculadas:* contiene las puertas a calcular (nombre e identificador) y la operación a realizar para obtenerlas.

- *calc_operandos:* contiene para cada puerta a calcular los operandos a partir de los que se obtiene.

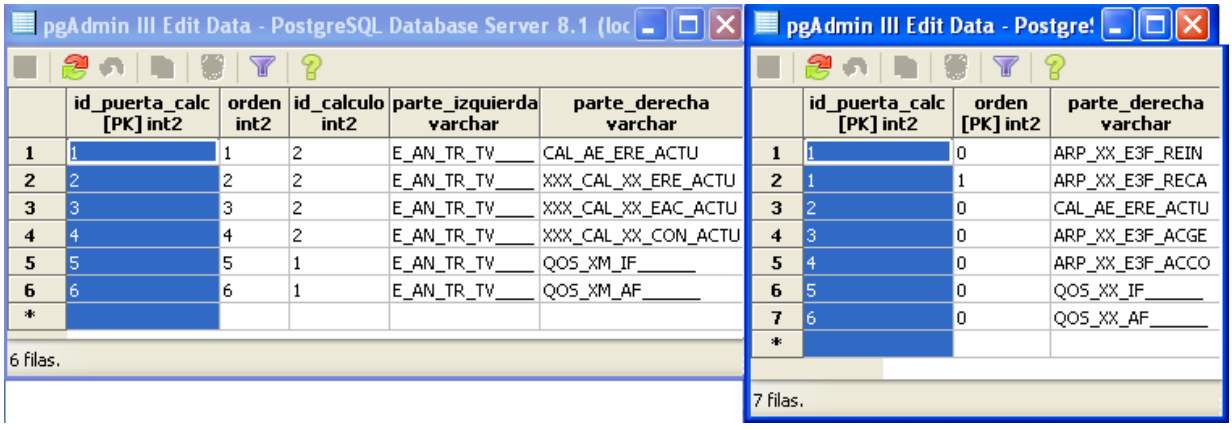


Imagen 56: Contenido de las tablas *calc_puertascalculadas* y *calc_operandos*

4.4. Módulo de generación de alarmas

El Modulo de generación de alarmas es un plugin del núcleo que permite al sistema detectar y notificar al usuario las alarmas originadas por diversas causas. Funciona siguiendo los mecanismos habituales, es decir, intercambia valores de variables con el resto del sistema a través del núcleo. Para su configuración, se recurre a una base de datos, en cuyas tablas se detallan los aspectos necesarios para funcionar en un entorno concreto. Entre éstos, se encuentran la lista de alarmas que se pueden generar y las condiciones que se darán para que ello ocurra.

Como resultado de su funcionamiento, interactúa con la base de datos, e inserta los datos referidos a las alarmas que vayan apareciendo, para que luego sean accesibles por otros módulos como, por ejemplo, los encargados de mostrar en pantalla dichas alarmas.

Es necesario que no se pierda el estado de las alarmas, debiendo estas mantenerse entre reinicios del módulo. De este modo se evitará que una misma alarma se notifique varias veces cada vez que se reinicia el módulo.

4.4.1. Alarmas

Una alarma es una señal que indica que ha ocurrido algo excepcional en el sistema. En el caso particular del módulo de generación de alarmas, la señal consiste en un mensaje textual, que va acompañado de una serie de atributos que facilitan su identificación y su gestión. Las alarmas tienen tres características:

- Denotan una situación anormal.
- Ocurren bajo ciertas condiciones, que pueden ser programadas por el operador.
- Deben ser reconocidas, bien por el usuario, o automáticamente.

Existe un conjunto de condiciones bajo las que se notifican alarmas, entre las que se enumeran las siguientes:

- **Generación de cyclics debidos a eventos programables**: se denomina evento programable a cada una de las circunstancias anómalas que pueden ser detectadas

por un ARP. Los analizadores pueden configurarse de forma que detecten estas situaciones y emitan una trama *cyclic*. En el Anexo C se incluyen todos los parámetros eléctricos sobre los que se pueden configurar eventos programables.

Un *cyclic* es un mensaje que se envía desde el ARP al núcleo, y que consiste en un volcado de los datos sobre las formas de onda y tensiones trifásicas que almacena el ARP en el momento de producirse la alarma. La finalidad de estos datos es proporcionar el contexto en el que se dio el evento que causó la alarma.

- **Fallo de comunicación**: las alarmas no se originan sólo en los ARPs, sino que también pueden ser producidas por el driver recomunicaciones del sistema de control central de parque, correspondiendo en este caso a un fallo de comunicaciones, en el que se distinguen dos variedades:
    - Fallo de comunicaciones global: corresponde al caso en el que es imposible comunicar con ningún analizador, debido a algún problema local, ajeno a la instalación en campo, por ejemplo, un fallo de la interfaz de red.
    - Fallo de comunicaciones de elemento ARP: este fallo ocurre en el caso en que el sistema no se encuentre en un fallo de comunicaciones global y sea incapaz de establecer comunicación con un analizador en concreto.

4.4.2. Funcionamiento del módulo

El módulo de generación de alertas trabaja como una serie de montaje. Cada vez que una puerta susceptible de generar alarmas cambia, se cotejarla con cada una de las alarmas definidas para determinar si se dispara realmente. Cuando la puerta dispara una alarma, se debe registrar la ocurrencia de esta.

El módulo se divide en tres subsistemas diferentes:

- **Suscripción y obtención de valores**: su cometido es suscribirse a las puertas que se le indiquen mediante la configuración. Cuando recibe notificaciones de cambios del núcleo, las despacha para que sean procesadas.
- **Procesado de valores**: debe analizar si las puertas obtenidas desde el módulo de suscripción disparan alarmas, en cuyo caso se pasarán al registro de alarmas.

- **Registro de Alarmas**: se encarga de la notificación al resto de módulos de la alarma producida y del almacenaje de los datos referentes a las alarmas que se vayan generando en la base de datos.

El diagrama UML del módulo queda recogido en la imagen 56:

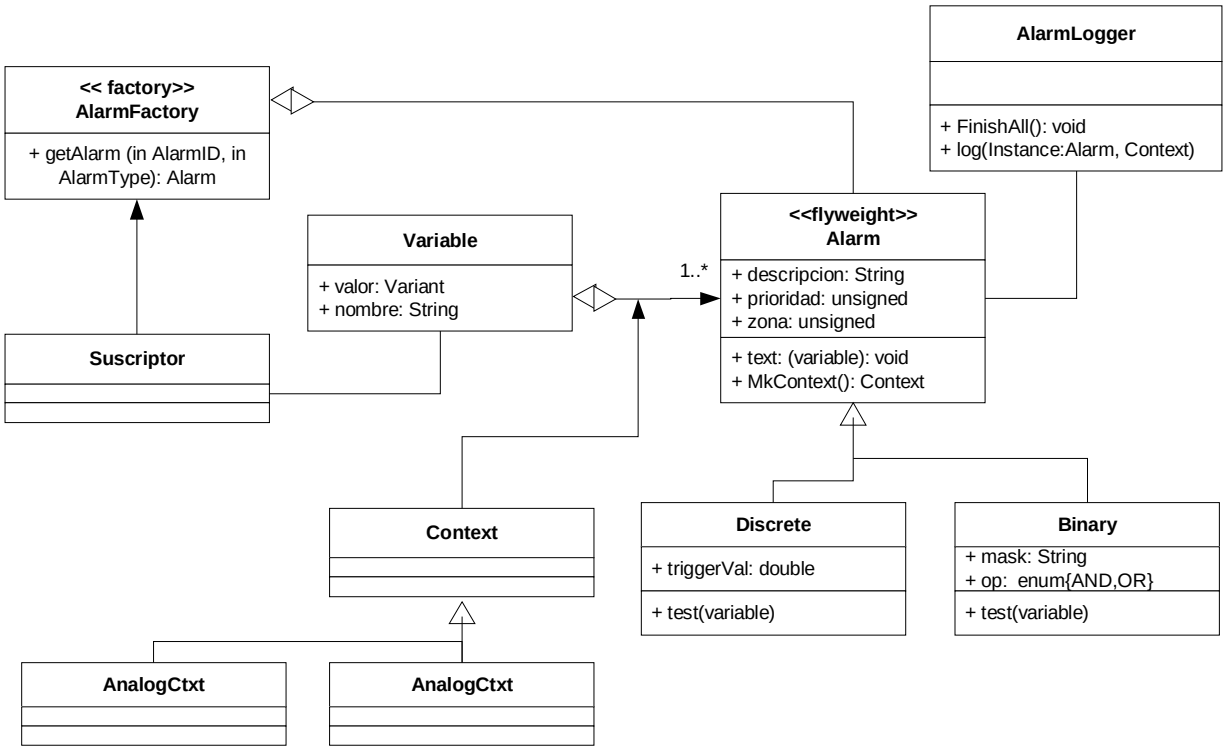


Imagen 56: Diagrama UML del módulo de generación de alarmas

- **Clase *Alarma***: objeto que recoge el comportamiento asociado a la comprobación del disparo de una alarma. Posee un método *text()* que toma como parámetros un objeto tipo *variable* y otro *context* y determina si debe dispararse la alarma. Tiene además un método *MkContext()* que genera objetos *context* específicos para el tipo de alarma indicado.

  Se definen dos tipos de alarmas:

  - Discreta: compara un dato analógico con un valor umbral *trigger*. Si este se sobrepasa (por encima o por debajo, según se especifique), se activa la alarma.

- Binaria: Realiza la operación *op* sobre la máscara *mask* y el valor binario de la puerta. Si el resultado es positivo, se activará la alarma.

- **Clase *Variable***: esta clase contiene la información relativa a una de las puertas susceptibles de generar una alarma.

  A cada objeto *variable* se le asocia una lista de las alarmas que puede disparar, así como el contexto de cada una de estas alarmas.

  Cuando se le indica, la variable recorre su lista de alarmas comprobando si debe activar alguna y refrescando sus contextos mediante la ejecución del método *test*().

- **Clase *Context***: esta clase recoge información extrínseca al tratamiento de las alarmas. Permite que se comparta un pequeño número de objetos *alarma* entre gran cantidad de puertas, ya que los aspectos dependientes de la relación de una puerta concreta con una alarma, se extraen y se almacenan en un objeto *contexto* propio del tipo de alarma. Puede darse el caso de que no haya información específica del tipo de alarma.

  En cualquier caso, hay que mantener siempre el estado de activación de la alarma, la fecha de inicio de ésta, y un identificador que distinga la instancia de la alarma de otras generadas por el mismo tipo.

- **Clase Subscriptor**: se encarga de las siguientes tareas:

  - Inicialización de los mapas de alarmas: crea un objeto *alarma* del tipo correspondiente (analógica o binaria) por cada alarma definida y carga los valores de los atributos necesarios para ese tipo de alarma. Esto se hará mediante la notificación a *AlarmFactory*.

  - Inicialización de los mapas de variables: se recogen las variables sobre las que se define alguna alarma. Para cada una se hace una suscripción al núcleo y se crea un mapa en el que aparecen los objetos tipo *alarma* y *context* asociados a dicha puerta.

  - Gestión de las variables notificadas: cuando el núcleo notifica el cambio de una variable, el *suscriptor* tiene la responsabilidad de indicar al objeto *variable*

asociado a dicha puerta que revise sus alarmas asociadas, de modo que se puedan disparar si se dan las condiciones apropiadas.

- **Clase *AlarmFactory***: recoge los detalles referentes a la creación de nuevas alarmas, para poder asociarlas a las variables. Cuando el objeto suscriptor lo solicita, *AlarmFactory* debe devolverle un puntero a un objeto *alarma*; si no existe lo crea y si exite, le envíe una referencia. Para poder saber qué alarmas se han disparado, se utilizará su identificador asignado en la base de datos.

  Posee un método *getAlarm()* que permite indicarle que debe cargar las alarmas desde la base de datos. Este método será invocado por el objeto *suscriptor*.

- **Clase Alarm*Logger:*** este objeto tiene como función materializar la notificación de las alarmas almacenándolas en la base de datos. Para ello posee un método *log()* que toma como parámetros un objeto tipo *alarma* y su *context*, y se encarga de extraer de ellos la información necesaria y volcarla en la base de datos:

  - Fechas de inicio y fin. Indican el peridoto durante el que la alarma estuvo activa. Caso de seguir activa, no habrá fecha de final.
  - Descripción de la alarma.
  - Identificador de la alarma: necesario para permitir localizar una alarma ya registrada para poder cambiar su estado a terminada, indicando una fecha de fin. *AlarmLogger*, como parte de su inicialización, debe asegurarse de que todas las alarmas registradas en sesiones anteriores aparezcan como terminadas.
  - Fecha de Reconocimiento: indica el momento en que ha sido reconocida, por el sistema de control o por un operador.
  - Estado de la alarma: valor booleano que indica si la alarma está activa o no.
  - Modo de finalización: determina si la alarma termina automáticamente, en el mismo instante que aparece o no. Cuando la alarma tiene auto-finalizar, no se tienen en cuenta
  - Severidad: indica la prioridad de la alarma.

- Zona: la zona indica dónde se ha dado la alarma.

El funcionamiento del módulo comienza con el arranque del sistema. Durante la fase de suscripción se genera un objeto tipo *alarma* por cada alarma definida. A continuación se recuperan de la base de datos aquellas puertas que pueden lanzar una alarma, creando por cada una de ellas un objeto *variable* y un mapa en el que se le asocian los objetos *alarma* y *context* que definen el comportamiento de las alarmas que lleva asociada dicha puerta. Para finalizar esta fase, se realizará la suscripción en el núcleo de todas las puertas a monitorizar.

A partir de ese momento, cada vez que el núcleo notifica un cambio en un valor de una de las puertas subscritas, se recorre el mapa con las alarmas asociadas a cada dicha puerta y se ejecuta el método *text()* de cada una, que comprueba si hay que lanzar la alarma o no y actualiza el contexto asociado.

Caso de que la alarma deba activarse, se llama a la función *log()*, pasándole como parámetros el objeto tipo alarma y el contexto. Ésta se encarga de obtener la información necesaria y generar y ejecutar la consulta para insertar la información en la base de datos.

4.4.3. Configuración del módulo

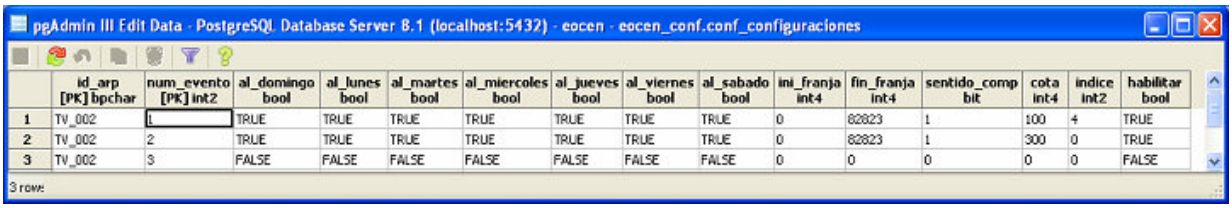
La configuración del módulo se carga desde las tablas y vistas del esquema *eocen_conf.* Los parámetros que se pueden configurar son:

- **Eventos programables:** desde la página de configuración de eventos del SCWEOCEN, el operador puede aplicar hasta tres disparadores de eventos por cada analizador de red, especificando para cada uno el parámetro eléctrico sobre el que se aplica, la cota y el sentido de la comparación, y el intervalo temporal durante el que el disparador estará activo.

  Una vez validada, la configuración es enviada por el driver de comunicaciones al ARP correspondiente y queda almacenada en la tabla eocen_conf.conf_configuraciones de la base de datos.

- **Puertas**: la configuración de las puertas sobre las que se defininen alarmas, así como el tipo de alarmas y los parámetros de configuración de las mismas está recogida en la vista *eocen_conf.conf_alarmas* y se obtiene de los datos introducidos en las tablas

conf_alarmas_patrones, que contiene los patrones de las puertas sobre los que se aplicacan las alarmas (utilizados para la subscripción en el núcleo), def_alarmas_tipo, que contiene los tipos de alarmas: binaria y cyclic o discreta, y *def_alarmas*, que contiene, para cada tipo de alarma definida en la tabal anterior, los parámetros que configuran su comportamiento.


pgAdmin III Edit Data - PostgreSQL Database Server 8.1 (localhost:5432) - eocen - eocen_conf.conf_configuraciones

	id_arp [PK] bpchar	num_evento [PK] int2	al_domingo bool	al_lunes bool	al_martes bool	al_miercoles bool	al_jueves bool	al_viernes bool	al_sabado bool	ini_franja int4	fin_franja int4	sentido_comp bit	cota int4	indice int2	habilitar bool
1	TV_002	1	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	0	82823	1	100	4	TRUE
2	TV_002	2	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE	0	82823	1	300	0	TRUE
3	TV_002	3	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0	0	0	0	0	FALSE

3 rows

Imagen 57: Información sobre la configuración de eventos programables de la máquian 002 contenida en la tabla conf_configuraciones

4.5. Integración de todos los módulos

Una vez implementados todos los módulos por separado, y obtenidos los consiguientes ejecutables de cada uno, se pasó a la fase de integración de los mismos en el servidor principal del proyecto.

Como ya se ha comentado en los distintos apartados que detallan la implementación de los módulos, existe un único programa, el núcleo, siendo el resto de plugins librerías dinámicas (extensión *.so*) arrancadas por éste. Para que este procedimiento funcionase, fue necesario establecer una estructura de carpetas adecuada con las aplicaciones e indicarle al núcleo dónde encontrar cada uno de estos plugins. Para ello, se definió una carpeta */home/eocen/eocen_bin* como raíz de la estructura que contuviese a la aplicación núcleo y una subcarpeta *plugins* en la que se almacenasen todos los plugins.

```
mkdir /home/eocen/eocen_bin
cp nucleo /home/eocen/eocen_bin

mkdir /home/eocen/eocen_bin/plugins
cp ModuloAlarmas.so /home/eocen/eocen_bin/plugins
cp ModuloBombeoTV.so /home/eocen/eocen_bin/plugins
```

```
cp ModuloCalculo.so /home/eocen/eocen_bin/plugins
cp DriverEocen.so /home/eocen/eocen_bin/plugins
cp ServidorHttp.so /home/eocen/eocen_bin/plugins
```

A continuación, se generaron los ficheros de configuración (extensión *.cfg*) de cada uno de los módulos. Estos ficheros deben encontrarse en el mismo directorio que el módulo correspondiente y se cargan al iniciarse la aplicación. En ellos se incorporó un campo identificador que permitiese al núcleo discriminar entre los distintos plugins. Los ficheros de configuración son:

- **Nucleo.cfg**: fichero de configuración del núcleo. En el se establece el tipo e IP de la base de datos de la aplicación, junto con el nombre del usuario y contraseña para poder acceder a ella. Se especifica también la tabla que contiene todas las puertas de la aplicación para que el núcleo, al arrancar, pueda cargarla y, a partir de ella, generar las tablas de suscripción de los plugins.

```
-- nucleo.cfg: fichero de configuración del núcleo –

tipoBD=PostgreSQL
nombreBD=192.168.104.25:5432:eocen
usuario=eocen
password=eocen
tabla=eocen_conf.conf_puertas
columnaId=id_puerta
columnaNombre=nombre
```

- **DriverEocen.cfg**: fichero con la configuración del driver de comunicaciones. Contiene el identificador del plugin (aquel que utilizará el núcleo para referirse a él), los parámetros necesarios para acceder a la base de datos y las puertas a través de las que se recibirá la información sobre las órdenes a enviar. Finalmente, se añadió un parámetro *debug* que indica si se debe generar un fichero de log específico para este plugin.

```
PluginId=1982

DBServer=192.168.104.25:5432:eocen
DBUser=eocen
DBPass=eocen
Ordenes=PET_XX_XXX_TIPO;PET_XX_XXX_PAR;PET_CF;
```

```
Debug=ON
```

- **ModuloBombeoTV.cfg**: fichero de configuración del módulo de bombeo asociado al parque de Tahivilla (a diferencia del resto de plugins que poseen un único archivo de configuración, en el caso del módulo de bombeo, habría que insertar un fichero de configuración distinto por cada parque a bombear). En el se indica el identificador del plugin, la información necesaria para el acceso a la base de datos, las tablas de las que debe recoger la información sobre las puertas a bombear, el identificador del parque y el parámetro *debug*.

```
PluginID=6101

BDatos=192.168.104.25:5432:eocen
user=eocen
pass=eocen
tablaPatrones=eocen_conf.conf_bombeo_patrones
tablaPuertas=eocen_conf.conf_bombeo
id_bombeo=TV

debug=OFF
```

- **ModuloCalculo.cfg**: fichero de configuración del plugin de cálculo. Contiene el identificador del plugin, la información necesaria para acceder a la base de datos, las tablas de las que debe cargar los tipos de cálculos configurados, los operandos y las operaciones a realizar sobre cada uno, la periodicidad en segundos de las operaciones y el parámetro *debug.*

```
PluginID=9101

BDatos=192.168.104.25:5432:eocen
user=eocen
pass=eocen

tablacalculos=eocen_conf.Def_Calculos
tablapuertascalculadas=eocen_conf.Calc_PuertasCalculadas
tablaoperandos=eocen_conf.Calc_Operandos
tablapuertascompletas=eocen_conf.conf_puertascompletas
periodobarrido=4

debug=ON
```

- **ModuloAlarmas.cfg**: fichero de configuración del módulo de alarmas. Contiene el identificador del plugin, la información necesaria para acceder a la base de datos, las tablas de las que debe cargar su configuración, la lista de parámetros eléctricos que pueden generar una alarma y el parámetro *debug.*

```
PluginID=8101

BDatos=192.168.104.25:5432:eocen
user=eocen
pass=eocen
tablaConf=eocen_conf.conf_alarmas
tablaParametros=eocen_conf.conf_parametros

debug=ON
```

- **ServidorHttp.cfg**: fichero de configuración del plugin de recepción de ordenes. Contiene el identificador del plugin, el puerto por el que debe escuchar las órdenes y el parámetro *debug*.

```
PluginId=2005
puerto=2005
debug=ON
```

Una vez finalizada la configuración de los módulos, se creó el script de arranque del núcleo en la carpeta */etc/init.d/*. En el se introdujeron las ordenes necesarias para realizar un rotado de los logs antes de lanzar la aplicación haciendo uso de la función *logrotate.* Esta función comprime los ficheros de logs de la anterior ejecución de la aplicación y los almacena en la carpeta */home/eocen/eocen_bin/oldlogs* con la nomenclatura *nombre_Modulo.n.gz*, donde "n" es un numero secuencial, y a continuación borra el contenido de los ficheros log. Una vez finalizado este proceso, se exportan las variables de entorno necesarias para el funcionamiento del sistema y se lanza la aplicación

El contenido del script es:

```
-- rotado de logs
```

```
logrotate -f /etc/logrotate.d/nucleo
logrotate -f /etc/logrotate.d/DriverEOCEN
logrotate -f /etc/logrotate.d/ModuloBombeoTV
logrotate -f /etc/logrotate.d/ModuloAlarmas
logrotate -f /etc/logrotate.d/ModuloCalculo
logrotate -f /etc/logrotate.d/ServidorHTTP

-- exportación de las variables de entorno necesarias para el acceso a base de datos
. /home/oracle/oraexports

-- arranque del núcleo
cd /home/eocen/eocen_bin
/home/eocen/eocen_bin/nucleo  >> /home/eocen/eocen_bin/log.txt 1>> /home/eocen/e
ocen_bin/log.txt 2>> /home/eocen/eocen_bin/log.txt &
```

Tras generar el script, se le dieron permisos de ejecución:

```
chmod 111 /etc/init.d/nucleo
```

El siguiente paso fue descargar e instalar la última versión disponible de las librerías necesarias para el funcionamiento de los plugins:

- **Librerías necesarias para el servidor HTTP**:
    - libpcre (PCRE - Perl Compatible Regular Expressions), descargable desde la página del proyecto PCRE: http://www.pcre.org/.
    - libpme (PME (PCRE Made Easy - C++ PCRE Wrapper Library). Esta librería se puede obtener de la página http://xaxxon.slackworks.com/pme/.
    - libehs (EHS – Embebbed HTTP Server), que podemos encontrar en la página http://xaxxon.slackworks.com/ehs/.

  Tras descargar todas las librerías, se descomprimieron haciendo uso de la herramienta *tar* y se instalaron en el directorio */usr/local/lib*. Para ello se utilizaron los siguientes comandos:

```
tar -zxvf pcre-6.6.tar.gz
cd pcre-6.6
```

```
/pcre-6.6$ ./configure && make && make install


tar -zxvf pme-1.0.4.tar.gz
cd pme-1.4.0
/pme-1.0.4$ ./configure && make && make install


tar -zxvf ehs-1.4.0.tar.gz
cd ehs-1.4.0
/ehs-1.4.0$ ./configure && make && make install
```

Para que el plugin pueda cargar las librerías hubo que copiarlas al archivo */lib* y crear sus enlaces simbólicos. Esto se hizo con los siguientes comandos:

```
cd /usr/local/lib
cp libehs.so /lib
cp libpme.so /lib

cd /lib
ln –s libehs.so libehs.so.0
ln –s libpme.so libpme.so.0
```

- **Librerías y drivers necesarios para el acceso a PostgreSQL por ODBC:**
  - unixodbc
  - odbc-postgresql
  - odbcinst1

Todas estas librerías vienen incorporadas en la versión Ubuntu del sistema operativo, por lo que sólo hubo que instalarlas mediante el uso de los comandos *apt*:

```
apt –get install unixodbc
apt –get install odbc-postgresql
apt –get install odbcinst1
```

Una vez instalados, se procedió a su configuración modificando dos ficheros:

- */etc/odbcinst.ini:* Este fichero sirve para registrar en unixodbc los driver ODBC instalados en el sistema. Para registrar el driver de PostgreSQL se le añadieron al fichero las siguientes las siguientes líneas:

```
[DriverPostgreSQL]
Description=PostgreSQL ODBC driver for Linux and Windows
Driver=/usr/lib/postgresql/lib/psqlodbc.so
Setup=/usr/lib/odbc/libodbcpsqlS.so
Debug = 0
CommLog = 1
```

- */etc/odbc.ini:* Este fichero sirve para registrar la configuración del servidor de la base de datos. Se le añadieron las siguientes líneas:

```
[EOCEN]
Description = PostgreSQL template1
Driver = DriverPostgreSQL
Trace = No
TraceFile = /tmp/odbc.log
Database = eocen
Servername = 192.168.104.247
UserName = postgres
Password = postgres
Port = 5432
Protocol = 6.4
ReadOnly = Yes
RowVersioning = No
SHowSystemTables = No
ShowOidColumn = No
FakeOidIndex = No
```

Finalmente, tras instalar y configurar las librerías necesarias para los plugins, se editó el archivo de configuración de la función *logrotate, /etc/logrotate.d*. Para su correcto funcionamiento, necesita de una entrada en el fichero de configuración por cada plugin y otra para el núcleo. El contenido de las mismas es:

```
/home/eocen/eocen_bin/logs/nombre_log.log {
        rotate 100
        missingok
        size=10000k
        olddir /home/eocen/eocen_bin/oldlogs
        copytruncate
```

```
        compress
}
```

Cabe reseñar que la aplicación, al arrancarse, genera varios logs, uno de ellos es el correspondiente a la ejecución del núcleo, el *log.txt,* y se almacena en el directorio raíz de la aplicación. Los restantes se corresponden con aquellos módulos que tengan activada la depuración (parámetro *debug* del fichero de configuración) y se almacenan en la carpeta */home/eocen/eocen_bin/logs* con la nomenclatura *nombre_plugin.log*.

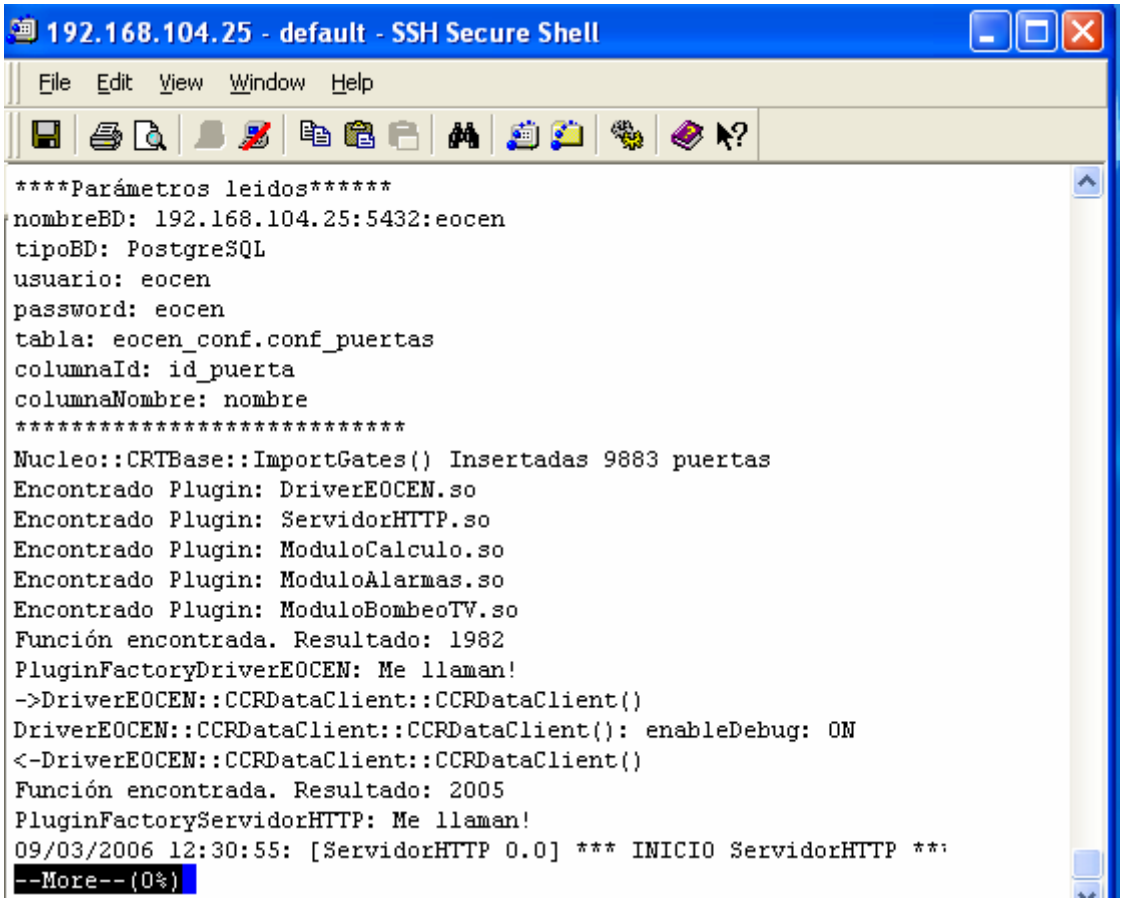


Imagen 58: Contenido del archivo de log del núcleo al arrancar

Anexo A: Disposiciones del Real Decreto 436/2004 sobre la energía eólica [1]

Nota: En el Real decreto, la clasificación de productores en régimen especial incluye a las energías eólicas con la siguiente nomenclatura:

- Grupo b.2 Instalaciones que únicamente utilicen como energía primaria la energía eólica. Dicho grupo se divide en dos subgrupos:
  - Subgrupo b.2.1 Instalaciones eólicas ubicadas en tierra.
  - Subgrupo b.2.2 Instalaciones eólicas ubicadas en el mar.

A1 Sobre la Calidad de la energía eólica

CAPÍTULO III.
CONDICIONES DE ENTREGA DE LA ENERGÍA ELÉCTRICA PRODUCIDA EN RÉGIMEN ESPECIAL.

Artículo 19. Obligaciones de los productores en régimen especial.

Sin perjuicio de lo establecido en el artículo 30.1 de la Ley 54/1997, de 27 de noviembre, los titulares de instalaciones de producción en régimen especial tendrán las siguientes obligaciones:

a. Entregar y recibir la energía en condiciones técnicas adecuadas, de forma que no se causen trastornos en el normal funcionamiento del sistema.

b. Abstenerse de ceder a consumidores finales los excedentes de energía eléctrica no consumida, excepto en el caso de que actúe de acuerdo con lo establecido en el artículo 22.1.b. No tendrán la consideración de cesión a consumidores finales, a estos efectos, los autoconsumos, es decir, la cesión que se realice a otro centro de la misma empresa, a sus filiales, matrices o a cualquiera de los miembros de una agrupación titular de la instalación, que constituyen un autoproductor tal como se define en el artículo 2.

c. Satisfacer los peajes y tarifas de acceso por la utilización de las redes de transporte o distribución cuando actúen como consumidores cualificados y celebren contratos de suministro de energía eléctrica.

d. Todas las instalaciones con potencias superiores a 10 MW deberán comunicar a la distribuidora una previsión de la energía eléctrica a ceder a la red en cada uno de los

períodos de programación del mercado de producción de energía eléctrica. Deberán comunicarse las previsiones de los 24 períodos de cada día con, al menos, 30 horas de antelación respecto al inicio de dicho día. Asimismo, podrán formular correcciones a dicho programa con una antelación de una hora al inicio de cada mercado intradiario.

Si las instalaciones estuvieran conectadas a la red de transporte, deberán comunicar dichas previsiones, además de al distribuidor correspondiente, al operador del sistema.

Estarán exentos de realizar todas estas comunicaciones aquellas instalaciones que opten por vender su energía eléctrica libremente en el mercado.

DISPOSICIÓN TRANSITORIA TERCERA. Conexión a la red.

1. En tanto el Ministerio de Economía no establezca nuevas normas técnicas para la conexión a la red eléctrica de estas instalaciones, continúa en vigor la Orden del Ministerio de Industria y Energía, de 5 de septiembre de 1985.

Asimismo, hasta entonces, deberán observarse los criterios siguientes:

a. Los titulares que no tengan interconectados en paralelo sus grupos con la red tendrán todas sus instalaciones receptoras o sólo parte de ellas conectables por un sistema de conmutación, bien a la red de la empresa distribuidora bien a sus grupos generadores, que asegurará que en ningún caso puedan quedar sus grupos generadores conectados a dicha red.

b. Los titulares que tengan interconectados en paralelo sus grupos con la red general lo estarán en un solo punto, salvo circunstancias especiales debidamente justificadas y autorizadas por la Administración competente, y podrán emplear generadores síncronos o asíncronos.

   Las instalaciones de potencia superior a cinco MW dotadas de generadores síncronos, en caso que la instalación ceda excedentes eléctricos a la red, deberán estar equipadas con sistemas de desconexión automática que eviten provocar oscilaciones de tensión o frecuencia superiores a las reglamentarias y averías o alteraciones en el servicio de la red.

   Estos titulares deberán cortar la conexión con la red de la empresa distribuidora si, por causas de fuerza mayor u otras debidamente justificadas y aceptadas por la Administración competente, la empresa distribuidora lo solicita. Las condiciones del servicio normal deberán, sin embargo, ser restablecidas lo más rápidamente posible. Cuando se dé esa circunstancia se informará al órgano competente.

c. La energía suministrada a la red deberá tener un cos ø lo más próximo posible a la unidad. Los titulares conectados en paralelo con la red deberán tomar las medidas necesarias para ello o llegar a acuerdos con las empresas distribuidoras sobre este punto.

A los efectos de este Real Decreto y para el cálculo del cos φ, se tomará la energía reactiva demandada cuando se entrega energía activa a la red.

d. En relación con la potencia máxima admisible en la interconexión de una instalación de producción en régimen especial, se tendrán en cuenta los siguientes criterios, según se realice la conexión con la distribuidora a una línea o directamente a una subestación:

    1. Líneas: la potencia total de la instalación conectada a la línea no superará el 50 % de la capacidad de la línea en el punto de conexión, definida como la capacidad térmica de diseño de la línea en dicho punto.

    2. Subestaciones y centros de transformación (AT/BT): la potencia total de la instalación conectada a una subestación o centro de transformación no superará el 50 % de la capacidad de transformación instalada para ese nivel de tensión.

Las instalaciones del grupo b.1 tendrán normas específicas que se dictarán por los órganos que tengan atribuida la competencia siguiendo los criterios anteriormente relacionados.

A2 Sobre las primas y bonificaciones

CAPÍTULO IV.
RÉGIMEN ECONÓMICO.

SECCIÓN I. DISPOSICIONES GENERALES.

Artículo 26. Complemento por energía reactiva.

1. Toda instalación acogida al régimen especial, en virtud de la aplicación de este Real Decreto, independientemente de la opción de venta elegida en el artículo 22, recibirá un complemento por energía reactiva. Este complemento se fija como un porcentaje de la tarifa eléctrica media o de referencia de cada año definida en el artículo 2 del Real Decreto 1432/2002, de 27 de diciembre, y publicada en el Real Decreto por el que se establece la tarifa eléctrica, en función de la categoría, grupo y subgrupo al que pertenezca la instalación, según se establece en la sección III de este capítulo IV, y del período horario en el que se entregue la energía. Dicho porcentaje se establece en el anexo V.

2. Este complemento será facturado y liquidado a la empresa distribuidora de acuerdo a lo establecido en los artículos 17 y 27.

3. Sin perjuicio de lo anterior, las instalaciones que opten por vender su energía en el mercado, según el artículo 22.1 b, podrán renunciar al complemento por energía reactiva establecido en este artículo, y podrán participar voluntariamente en el procedimiento de operación de control de tensión vigente, aplicando sus mecanismos de retribución.

SECCIÓN III. TARIFAS, PRIMAS E INCENTIVOS POR PARTICIPAR EN EL MERCADO.

Artículo 34. Tarifas, primas e incentivos para instalaciones de la categoría b, grupo b.2: energía eólica.

1. Instalaciones del subgrupo b.2.1 de no más de 5 MW de potencia instalada:

- Tarifa: 90 % durante los primeros 15 años desde su puesta en marcha y 80 % a partir de entonces.
- Prima: 40 %.
- Incentivo: 10 %.

2. Resto de instalaciones del subgrupo b.2.1:

- Tarifa: 90 % durante los primeros cinco años desde su puesta en marcha, 85 % durante los 10 años siguientes y 80 % a partir de entonces.
- Prima: 40 %.
- Incentivo: 10 %.

3. Instalaciones del subgrupo b.2.2 de no más de 5 MW de potencia instalada:

- Tarifa: 90 % durante los primeros 15 años desde su puesta en marcha y 80 % a partir de entonces.
- Prima: 40 %.
- Incentivo: 10 %.

4. Resto de instalaciones del subgrupo b.2.2:

- Tarifa: 90 % durante los primeros cinco años desde su puesta en marcha, 85 % durante los 10 años siguientes y 80 % a partir de entonces.
- Prima: 40 %.
- Incentivo: 10 %.

5. Sin perjuicio de lo dispuesto en el artículo 40, cuando el grupo b.2 alcance los 13000 MW de potencia instalada se procederá a la revisión de la cuantía de las tarifas, incentivos y primas expresadas en este artículo.

DISPOSICIÓN ADICIONAL CUARTA. Complemento por continuidad de suministro frente a huecos de tensión.

Aquellas instalaciones eólicas acogidas al grupo b.2, que cuenten con los equipos técnicos necesarios para contribuir a la continuidad de suministro frente a huecos de tensión, incluyendo

la oportuna coordinación de protecciones, tendrán derecho a percibir un complemento específico durante cuatro años.

Este complemento será equivalente al 5 % de la tarifa eléctrica media o de referencia de cada año definida en el artículo 2 del Real Decreto 1432/2002, de 27 de diciembre, independientemente de la opción de venta elegida en el artículo 22 de este Real Decreto.

Dicho complemento será aplicable únicamente a las instalaciones eólicas que presenten ante la empresa distribuidora y ante la Dirección General de Política Energética y Minas un certificado del fabricante donde se demuestre que se ha instalado esta mejora de operación.

La Dirección General de Política Energética y Minas tomará nota de esta mejora en la inscripción del Registro administrativo de instalaciones de producción de energía eléctrica y la comunicará a la Comisión Nacional de Energía, a los efectos de liquidación de las energías.

Este complemento será facturado y liquidado a la empresa distribuidora de acuerdo a lo establecido en los artículos 17 y 27.

El operador del sistema deberá proponer un procedimiento de operación en el que se regulen los requerimientos mínimos que han de cumplir las protecciones de las distintas instalaciones y tecnologías de producción en régimen especial, a efectos de garantizar la continuidad de suministro frente a huecos de tensión, estableciéndose asimismo un procedimiento transitorio para la adaptación de las instalaciones existentes.

ANEXO V.

Complemento por energía reactiva.

Tipo de FP	Energía activa y reactiva	Bonificación %		
	Factor de potencia	Punta	Llano	Valle
Inductivo.	< 0,95	- 4	- 4	8
	< 0,96 y ≥ 0,95	- 3	0	6
	< 0,97 y ≥ 0,96	- 2	0	4
	< 0,98 y ≥ 0,97	- 1	0	2
	< 1 y ≥ 0,98	0	2	0
	1	0	4	0
Capacitivo	< 1 y ≥ 0,98	0	2	0
	< 0,98 y ≥ 0,97	2	0	- 1
	< 0,97 y ≥ 0,96	4	0	- 2
	< 0,96 y ≥ 0,95	6	0	- 3
	< 0,95	8	- 4	- 4

Tabla 28: Complementos por energía reactiva fijados por el RD

El factor de potencia FP se obtendrá haciendo uso del equipo de medida contador-registrador de la instalación. Se calculará con dos cifras decimales y el redondeo se hará por defecto o por exceso, según que la tercera cifra decimal sea o no menor de cinco. Deberá mantenerse cada cuarto de hora, en el punto de conexión de la instalación con la red, dentro de los períodos horarios de punta, llano y valle del tipo tres de discriminación horaria, de acuerdo con el apartado 7.1 del anexo I de la Orden Ministerial de 12 de enero de 1995.

Los porcentajes de complemento se aplicarán con periodicidad cuarto-horaria, realizándose, al finalizar cada mes, un cómputo del acumulado mensual, que será facturado y liquidado según corresponda.

ANEXO VI.

Primas de las instalaciones a las que se refiere la disposición transitoria segunda de este Real Decreto.

Primas

Grupo	Tipo Instalación	Potencia (MW)	Prima (Cent €/kWh)
a.	a.1 y a.2	P ≤10	3,2424
b.	b.2		2,7500
	b.3		3,0373
	b.4		3,0373
	b.6		3,4224
	b.7		2,5970
c.		P ≤10	1,8244
Artículo 31			0,4935
d.	d.1		3,2424
	d.2		2,3729
	d.3		1,5180

Tabla 29: Primas fijadas en el RD

Anexo B: Generación de estudios de calidad [10]

Los estudios de calidad están formados por las medidas, en tantos por 10.000, de las líneas espectrales correspondientes a los 50 primeros armónicos, junto con 15 interarmónicos asociados a cada uno de los armónicos, de las tres tensiones trifásicas.

Las medidas son realizadas con el analizador de red principal (ARP), según la norma de medida UNE EN 61000-4-7 de noviembre de 1996, y posteriormente enviadas al centro de gestión de parque.

- **Parámetros temporales**: el tiempo de observación se calcula como el intervalo temporal real de medida más los intervalos entre medidas.

  Según la norma CEI 61000-3-6, el periodo mínimo de estudio será de una semana. De esta manera, partiendo de los intervalos denominados *muy cortos* (3 segundos), se harán promedios para establecer periodos *cortos* (10 minutos) reales y obtener un estudio *diezminutal*. Para hacer estudios diarios o semanales, se tomarán un número razonable de estudios muy cortos y/o cortos.

- **Procedimiento de cálculo**: para cada ventana temporal se calculan armónicos e interarmónicos y se toman valores promedio como armónicos e interarmónicos del intervalo de 3 segundos. A partir de ellos se obtienen poblaciones de armónicos y THD's de las muestras distribuidas a lo largo del periodo dura el experimento. Dichas muestras se emplearán tanto para hacer un estudio estadístico como para observar la evolución temporal de las mismas.

  Los valores U_{RMS_10min} se hallarán como promedios estadísticos de los valores U_{RMS_3s}, que son adquiridos por el SCADA al solicitar el informe de calidad al ARP.

  De forma idéntica se actuaría para el resto de intervalos, sabiendo que el número de intervalos de duración inferior es a elección.

B1 Normativa aplicada

- UNE-EN 61000-4-7. Compatibilidad electromagnética (CEM). Parte 4: Técnicas de ensayo y medida. Sección 7: Guía general relativa a las medidas de armónicos e interarmónicos, así como a los aparatos de medida, aplicable a las redes de alimentación y a los aparatos conectados a éstas. *Esta norma se emplea para establecer las condiciones de funcionamiento y modo de operación del aparato de medida.*

- CEI 61000-3-6. Electromagnetic compatibility (EMC). Part 3: Limits. Section 6: Assessment of emission limits for distorting loads in MV and HV power systems- Basic EMC publication, Technical report-type 3. *Primera edición, 1996-10. Norma empleada para establecer los límites de los armónicos.*

- UNE-EN 50160. Características de la tensión suministrada por las redes generales de distribución. Norma empleada para establecer los límites de distorsión armónica (THD) (coincide con la CEI 61000-3-6). En esta norma además se establece que en condiciones normales de explotación, durante cada periodo de una semana, el 95% de los armónicos deben permanecer por debajo de los límites que se establecen en esta norma, que respecto a la CEI 61000-3-6 son más permisivos (límites para armónicos 12, 14, 15, 16, 18, 20, 21, 22 y 24 se fijan en 0.5%, el resto igual -25 en adelante excluidos).

- UNE-EN 61000-2-4. Compatibilidad electromagnética (CEM). Parte 2: Entorno. Sección 4: Niveles de compatibilidad para las perturbaciones conducidas de baja frecuencia, en plantas industriales. *Esta norma se emplea para establecer los límites aconsejados para los interarmónicos (en entornos de clase 2)*.

B2 Estudio estadístico de armónicos e interarmónicos de tensión

Se empleará el percentil como unidad de medida. La norma CEI 61000-3-6, se centra en el percentil 95, aunque se pueden añadir en los informes cualesquiera otros.

La norma CEI 61000-3-6, impone 3 restricciones:

Los percentiles 95% de los armónicos hallados en los estudios de 3 segundos a lo largo de un día no excederán los valores de las tablas expuestas en dicha norma.

- El correspondiente en las tablas anteriores.
- El máximo semanal del percentil 95% de los estudios de 3 segundos no excederá de 1.5-2 veces el valor correspondiente en las tablas.

Queda claro, por tanto, que la base de los tratamientos estadísticos son los estudios de intervalos *muy cortos* (3 segundos) y *cortos* (10 minutos).

La forma en que se representan los datos es mediante tablas para los armónicos, en las que la columna de la izquierda indica los valores máximos de la norma CEI 1000-3-6. Los armónicos e interarmónicos se expresarán siempre como porcentaje del valor rms de la tensión de red. Estas tablas también se muestran de forma gráfica.

En cuanto a los interarmónicos se suele optar, exclusivamente, por una representación gráfica de los mismos. El valor aconsejado por la UNE-EN 61000-2-4 como cota para los interarmónicos es 0.2.

Tratamiento similar puede hacerse para la disposición de resultados de forma gráfica para los análisis semanales.

Partiendo de la primera condición se expondrá un ejemplo de la gestión estadística de resultados.

Armónico	Norma	Vr	Vs	Vt
1		99.9525	99.9518	99.9491
2	2	0.1456	0.1015	0.1378
3	5	0.1535	0.1226	0.1325
4	1	0.0738	0.0659	0.0643
5	6	0.6489	0.6443	0.7291
6	0.5	0.0468	0.0528	0.0443
7	5	0.3094	0.2546	0.2872
8	0.5	0.2806	0.2731	0.2719
9	1.5	0.1042	0.1628	0.1001
10	0.5	0.4548	0.4795	0.5252
11	3.5	0.9053	1.0046	1.1651
12	0.2	0.1034	0.0840	0.1014
13	3	0.4792	0.4872	0.4782
14	0.2	0.7571	0.6817	0.7598
15	0.3	0.2655	0.2792	0.2363
16	0.2	2.3030	2.2923	2.5166
17	2	1.5100	1.3096	1.4398

18	0.2	0.2693	0.3381	0.3141
19	1.5	0.6149	0.4847	0.4920
20	0.2	1.0941	1.3638	1.3405
21	0.2	0.2967	0.3085	0.3536
22	0.2	1.2816	1.3471	1.4224
23	1.5	0.4641	0.4911	0.4670
24	0.2	0.2178	0.2825	0.2226
25	1.5	0.4301	0.4303	0.4154
26	0.2	0.5129	0.3770	0.4778
27	0.2	0.1772	0.1658	0.1746
28	0.2	0.7369	0.8166	0.7712
29	1.3207	0.4435	0.4826	0.4289
30	0.2	0.1334	0.1721	0.1665
31	1.2484	0.2256	0.2355	0.2289
32	0.2	0.3558	0.2749	0.4080
33	0.2	0.0796	0.0809	0.0877
34	0.2	0.3111	0.3059	0.2975
35	1.1286	0.2032	0.2090	0.2211
36	0.2	0.0425	0.0425	0.0483
37	1.0784	0.1354	0.1407	0.1354
38	0.2	0.0919	0.0857	0.0897
39	0.2	0.0326	0.0387	0.0456
40	0.2	0.0729	0.0811	0.0819
41	0.9927	0.0821	0.1055	0.1157
42	0.2	0.0324	0.0340	0.0352
43	0.9558	0.0841	0.0860	0.0906
44	0.2	0.0639	0.0594	0.0588
45	0.2	0.0271	0.0341	0.0369
46	0.2	0.0355	0.0425	0.0362
47	0.8915	0.0875	0.0872	0.1067
48	0.2	0.0186	0.0220	0.0240
49	0.8633	0.0671	0.0661	0.0625
50	0.2	0.0362	0.0301	0.0348

Tabla 29. Valores máximos impuestos por la norma para cada uno de los armónicos junto con los valores de los armónicos percentil 95% empleados para el ejemplo

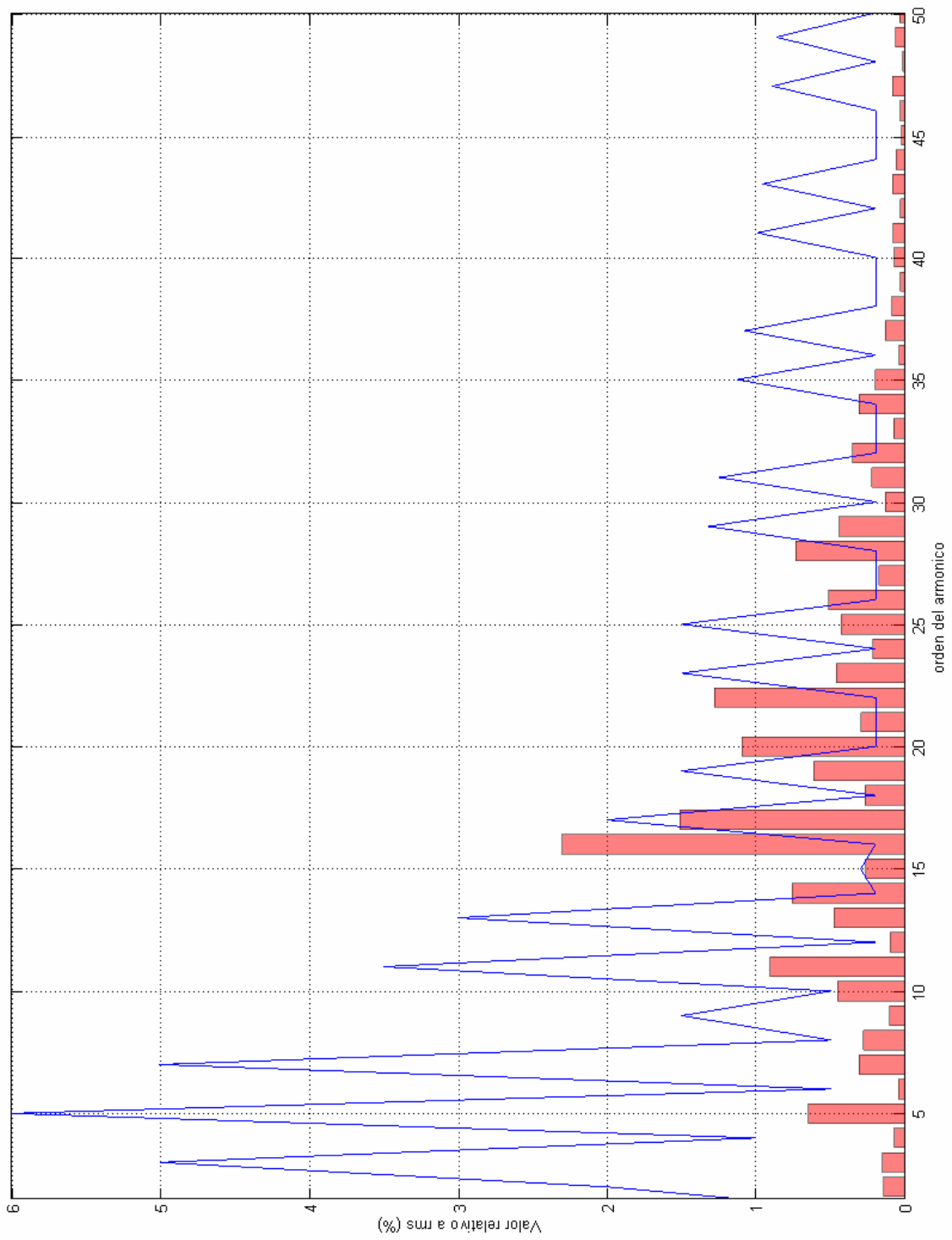


Imagen 59. Armónicos percentil 95% de Vr del ejemplo

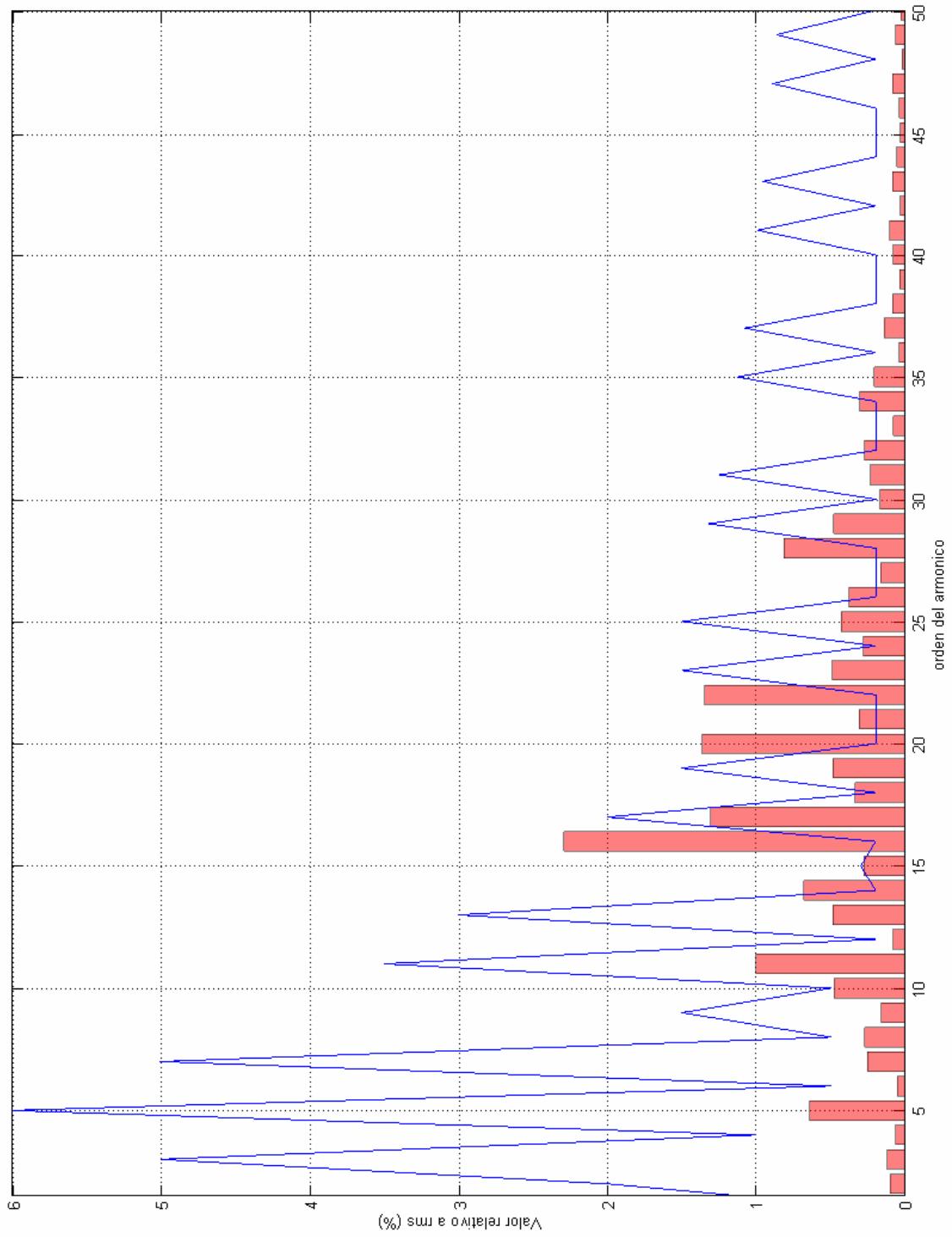


Imagen 60. Armónicos percentil 95% de Vs del ejemplo

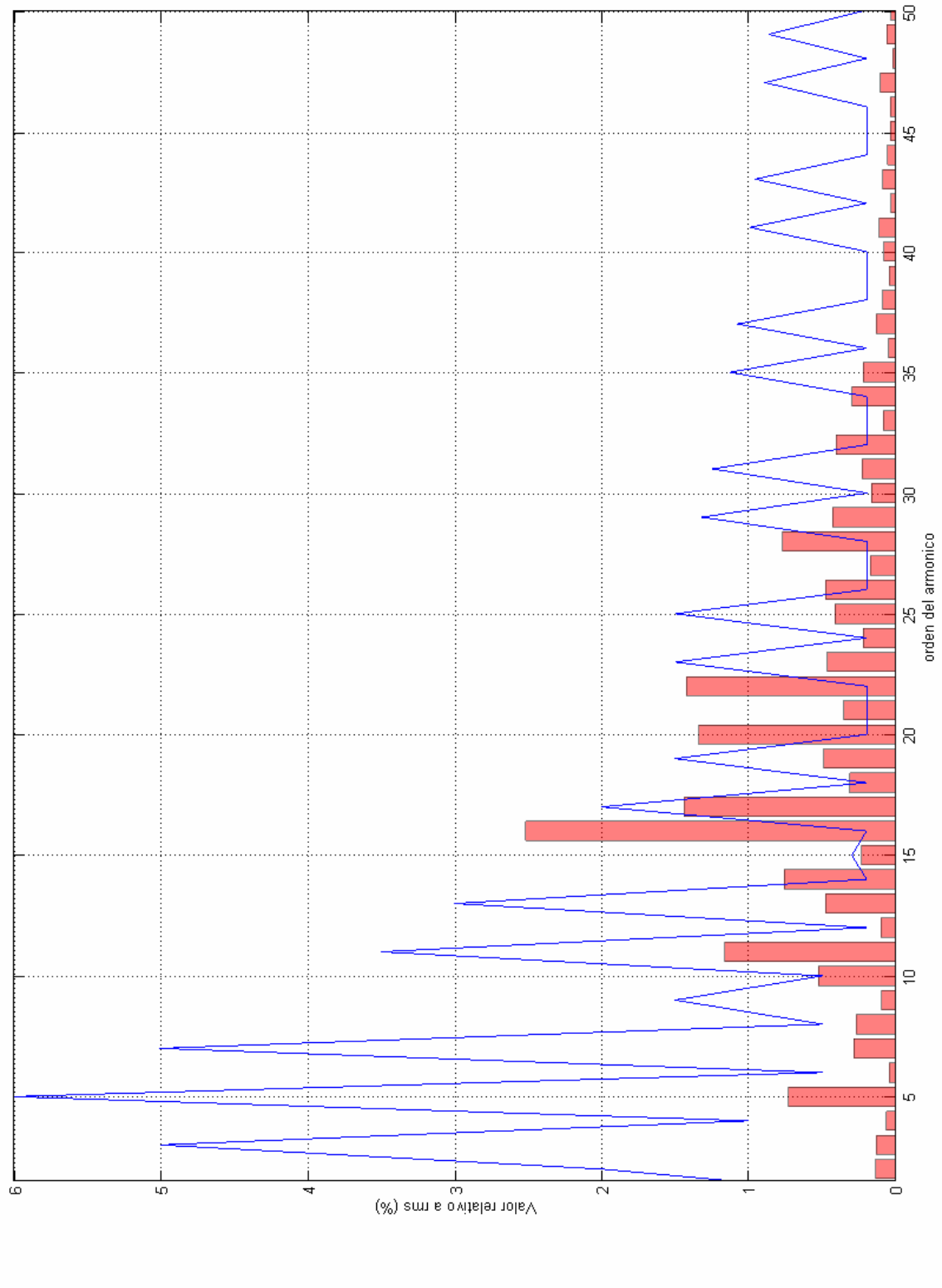


Imagen 61. Armónicos percentil 95% de Vt del ejemplo

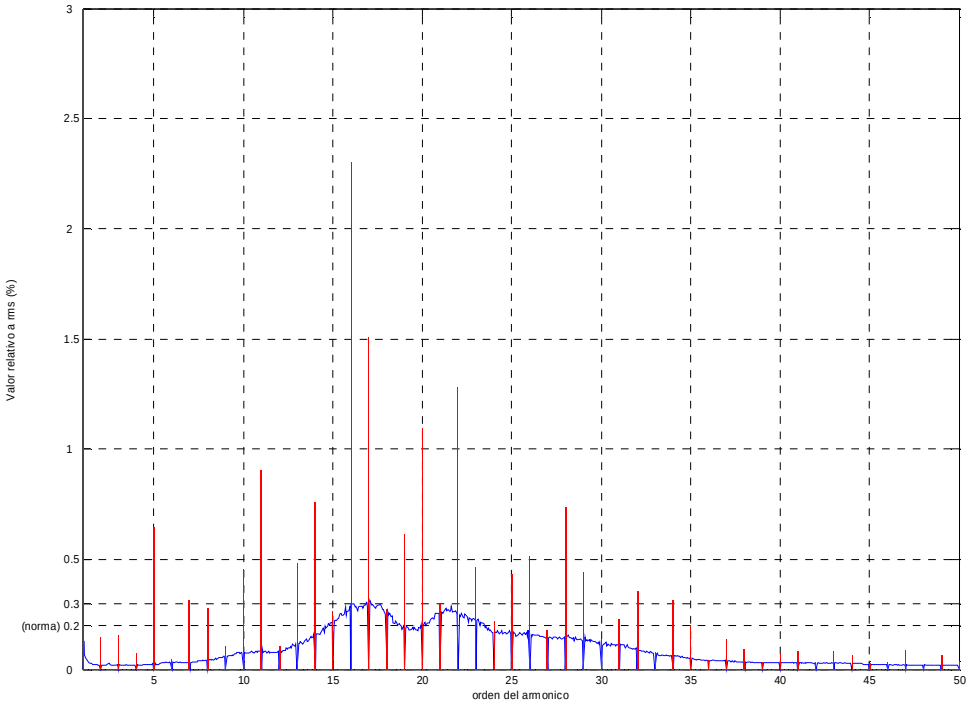


Imagen 62. Armónicos e interarmónicos percentil 95% de Vr

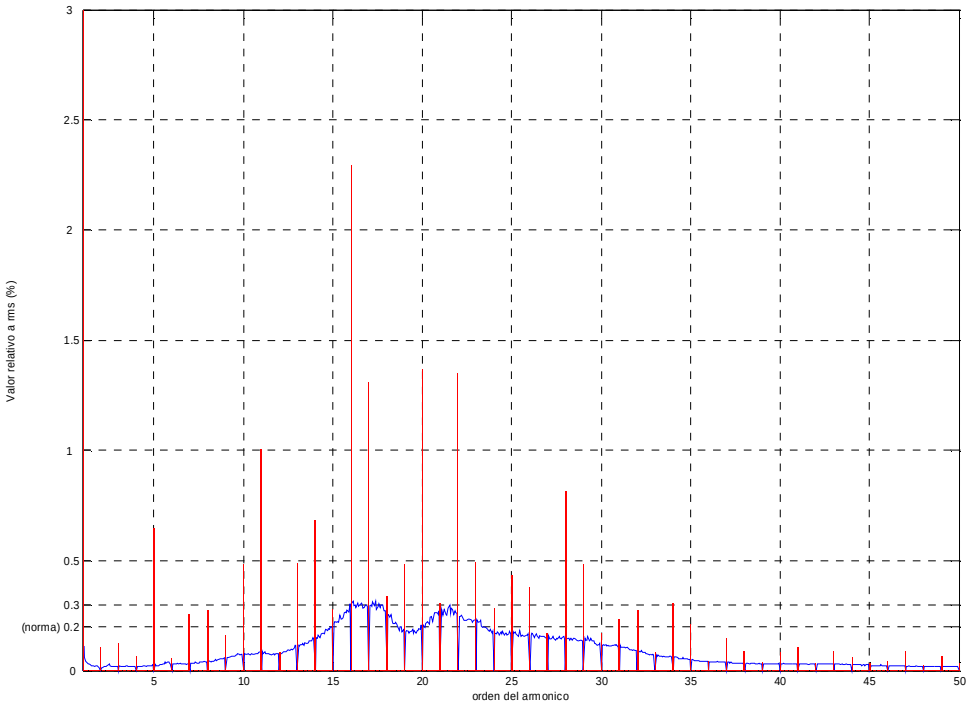


Imagen 63. Armónicos e interarmónicos percentil 95% de Vs

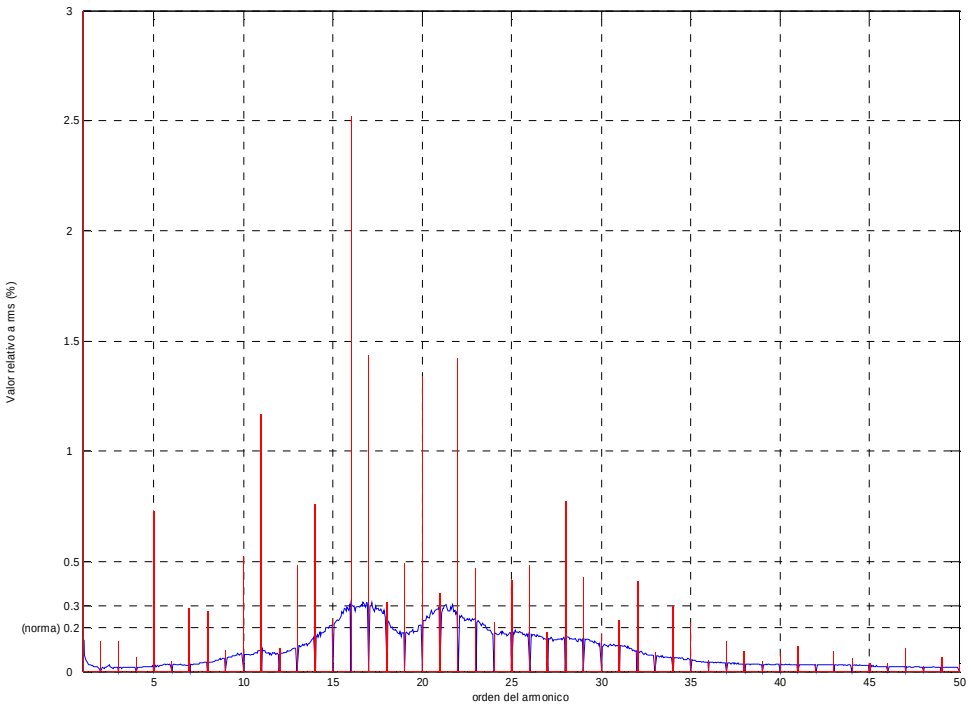


Imagen 64. Armónicos e interarmónicos percentil 95% de Vt

Anexo C: Tabla de parámetros electricos

Índice	Parámetro eléctrico	Unidad
0	Tensión fase 1	V
1	Tensión fase 2	V
2	Tensión fase 3	V
3	Tensión fase-neutro trifásico	V
4	Corriente fase 1	A
5	Corriente fase 2	A
6	Corriente fase 3	A
7	Corriente trifásico	A
8	Tensión fase 1-2	V
9	Tensión fase 2-3	V
10	Tensión fase 3-1	V
11	Tensión fase-fase trifásico	V
12	Potencia activa generada fase 1	KW
13	Potencia activa generada fase 2	KW
14	Potencia activa generada fase 3	KW
15	Potencia activa generada trifásico	KW
16	Potencia activa consumida fase 1	KW
17	Potencia activa consumida fase 2	KW
18	Potencia activa consumida fase 3	KW
19	Potencia activa consumida trifásico	KW
20	Potencia reactiva L fase 1	KVAr
21	Potencia reactiva L fase 2	KVAr
22	Potencia reactiva L fase 3	KVAr
23	Potencia reactiva L trifásico	KVAr
24	Potencia reactiva C fase 1	KVAr
25	Potencia reactiva C fase 2	KVAr
26	Potencia reactiva C fase 3	KVAr
27	Potencia reactiva C trifásico	KVAr
28	Potencia aparente fase 1	KVA
29	Potencia aparente fase 2	KVA
30	Potencia aparente fase 3	KVA
31	Potencia aparente trifásico	KVA
32	Factor de potencia fase 1	--
33	Factor de potencia fase 2	--
34	Factor de potencia fase 3	--
35	Factor de potencia trifásico	--
36	THD tensión fase 1	%
37	THD tensión fase 2	%
38	THD tensión fase 3	%
39	THD corriente fase 1	%
40	THD corriente fase 2	%
41	THD corriente fase 3	%
42	Frecuencia	Hz
43	Hueco	--
50	Petición por parte del PC	--