PROYECTO FIN DE CARRERA

Ingeniería Técnica Industrial: Electrónica

SCADA Online

Autor

Ricardo Garzo Castro

Director

Jesús Ponce de León

Escuela de Ingeniería y Arquitectura – Universidad de Zaragoza

Marzo 2014

.

Resumen

El presente proyecto consiste en un SCADA para la monitorización y supervisión de autómatas programables de forma remota con un enfoque al control de procesos industriales. El SCADA ha de ser capaz de acceder a cualquier PLC sin importar la marca del mismo. Además del control de autómatas, el SCADA también dispone de funcionalidades como pueden ser alarmas asociadas a eventos y el manejo de usuarios dentro de la plataforma.

Se ha diseñado utilizando Python como lenguaje de programación principal de la aplicación y adicionalmente JavaScript como lenguaje de programación en el interfaz de usuario, obteniendo así una aplicación web, accesible tanto para ordenadores como para móviles.

Durante el mes de Febrero y Marzo de 2014, la aplicación web estará a uso de los interesados en la siguiente dirección IP: *http://155.210.159.208:80/home* . La aplicación web se encuentra alojada en un servidor del laboratorio de audio y video del departamento de electrónica de la Escuela de Ingeniería y Arquitectura. Al servidor está conectado un autómata programable de Siemens además de un segundo autómata simulado por software, y ambos se podrán controlar. Los credenciales para acceder al sitio web serán:

- Usuario: usuario
- Contraseña: contraseña

Finalmente, señalar que el proyecto ha nacido de la Cátedra entre TAIM-WESER y la Universidad de Zaragoza.

Índice

Índice de Figuras

1. Capítulo 1. Introducción y objetivos.

La Cátedra **TAIM WESER – Universidad de Zaragoza** nace en 2010 con el propósito de generar proyectos de investigación avanzada y aplicada en el entorno de la ingeniería de alto nivel. Su objetivo es favorecer la Investigación, el Desarrollo y la Innovación (I+D+i) en tres perfiles estratégicos: ingeniería y bienes de equipo, tratamientos de residuos y, especialmente, energías renovables.

Para ello, se ha establecido una línea de colaboración permanente entre los técnicos de la empresa y los docentes universitarios de cara al desarrollo de una política de formación práctica de los estudiantes e investigadores. Se trata de una unión estratégica y duradera, por la que ambas partes se benefician de los resultados de la investigación, el desarrollo y la innovación.

En este marco, se apoya especialmente la realización de tesis doctorales y proyectos fin de carrera (PFC) de I+D+i con un alto contenido científico y tecnológico.

El primer proyecto de esta Cátedra, que es este proyecto fin de carrera, trata de la *creación de una plataforma remota multiacceso para mantenimiento y control de autómatas*.

1.1 Objetivos

TAIM WESER define un guión sobre el cual se llevará a cabo el proyecto, y donde se exponen los siguientes objetivos que se alcanzarán mediante la construcción de una plataforma software:

1. Supervisión y control de los estados de las variables de los PLCs y la reprogramación de sus tareas con independencia de la marca de los mismos.
2. Qué todo se pueda realizar de forma remota, vía Internet, y los usuarios que accedan tengan restricciones en función de su nivel de acceso.

En otras palabras, lo que TAIM WESER desea construir es un **SCADA**, acrónimo de Supervisory Control And Data Acquisition (control de supervisión y adquisición de datos). Una plataforma software que permita controlar y supervisar procesos industriales a través de internet.

Como objetivo secundario se pretenderá:

- Que el SCADA tendrá un interfaz gráfico intuitivo para el usuario y sea accesible desde distintas plataformas, ordenadores o dispositivos móviles, y además sea fácilmente expandible en el futuro.

1.2 Ideas previas a la realización del proyecto

Respecto a uno de los objetivos a alcanzar por la empresa, existe un problema. Que es el de reprogramar o incluir nuevas tareas en PLCs. El autor consideró que este objetivo no se puede lograr de una manera sencilla.

Desde un punto de vista técnico, si se quieren programar tareas en un PLC de forma remota, se necesita acceso a una herramienta software que pueda:

- Transformar el código de la tarea a instrucciones que entienda el PLC y alojarlo en la memoria del mismo.
- Que a esta herramienta de software se pueda controlar de forma remota.

Actualmente existen herramientas de software que cumplen la primera parte, son los programas de escritorio para la programación de PLC que proporciona cada fabricante de PLC. Pero desafortunadamente, son programas de escritorio con un entorno gráfico, lo que dificulta su control remoto.

Los fabricantes de PLCs tampoco proporcionan herramientas de desarrollo para crear algo que cumpla lo expuesto arriba, y por lo tanto no se ha podido encontrar y mucho menos fabricar alguna herramienta que pueda cumplir el objetivo.

Por otra parte, y centrándose en un aspecto del modo de operación, reprogramar tareas de PLCs sin estar físicamente presente en la planta donde se lleve a cabo el proceso presenta un peligro potencial muy alto. Es obvio que si durante la realización de la tarea del PLC algo perjudicial ocurriera no existiría forma manual de parar la tarea.

Finalmente, y con todo lo explicado, se abandona el objetivo de la reprogramación de tareas remota; tanto por razones de seguridad como razones técnicas.

Respecto a los demás objetivos, no existe ninguna limitación y serán llevados a cabo.

1.3 Organización del documento

Esta memoria está dividida en cinco capítulos que abarcan distintos aspectos sobre la realización de este proyecto fin de carrera.

1. Introducción: El presente capítulo que sitúa al lector en el marco de creación del proyecto.
2. Estado del arte: Un capítulo dedicado a los antecedentes históricos relacionados con este proyecto así como la terminología empleada en el mundo de los SCADA.
3. Desarrollo: el capítulo más extenso de este documento, pues explica la fase de creación y tecnologías involucradas en el presente proyecto además de las distintas funcionalidades empleadas en la creación del SCADA.
4. Resultados: donde se expondrá el proyecto creado.
5. Conclusiones.

2. Capítulos 2. Estado del arte

Con este capítulo se pretende situar al lector en el entorno que rodea a los SCADA, tanto en la terminología técnica que se usa para hablar de los elementos de construcción de un SCADA como el contexto tecnológico en el que están englobados.

2.1 Conceptos previos

Los siguientes términos deberían ser familiares para el lector. Si bien no es necesario conocerlos en detalle, es conveniente entender que significan:

- Aplicación web: Es una aplicación de software que se ejecuta en un navegador web.

- Plugin: Un complemento, o plugin, es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica.
- API: Interfaz de programación de aplicaciones o API, en inglés es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción
- Log: Un log es un registro de eventos durante un rango de tiempo en particular.
- Script: Es un guión o archivo de órdenes que forma un programa para un entorno especial y automatiza la ejecución de tareas.
- Framework: Es una estructura conceptual con módulos de software concretos que sirve de base para la organización y desarrollo de software.
- HTTP: el protocolo usado en cada transacción de la World Wide Web.

2.2 Antecedentes

Actualmente, por SCADA se entiende un software que permita controlar y supervisar procesos industriales a distancia y controlarlos automáticamente. Provee de toda la información que se genera en un proceso productivo. Esto está íntimamente ligado con el concepto de realimentación que es el proceso de observar procesos con la intención de recabar información para mejorar o modificar diversos aspectos del funcionamiento. Un SCADA no deja de ser un sistema de control industrial.

Los sistemas de control industrial son sistemas controlados por ordenador para monitorizar y controlar procesos industriales que existen en el mundo físico. Son usados generalmente en industrias eléctricas, de tratamiento de aguas, petróleo, gas y datos. Los sistemas de control industrial usan datos que reciben de estaciones remotas para llevar a cabo tareas de supervisión, ya sea de manera automatizada o mediante técnicos, sobre elementos de control. Estos elementos controlan operaciones locales como la apertura de válvulas y accionamientos de frenado, recolección de datos a partir de sensores y monitorización de variables del entorno local para condiciones de alarma.

Figura 1 Imagen de una planta manejada por un sistema de control industrial.

El término de sistema de control industrial es un concepto general que engloba varios tipos de sistemas de control como son los sistemas de supervisión y adquisición de datos (SCADA), sistemas de control distribuido (DCS) y los controladores lógicos programables (PLC).

Los PLCs son un reemplazo electrónico de los sistemas cableados de relés, pues estos no son elementos estables, son difíciles de re-configurar y es complicado detectar el origen de los fallos. Por otra parte, los PLCs tienden a usarse en controles binarios de gran velocidad con un gran conjunto de entradas y salidas.

Figura 2 PLC de la marca Siemens.

Los DCS son un diseño funcional para el sistema de control distribuido que existe dentro de una planta de procesos industriales. Y nacen de la necesidad de capturar datos y controlar los sistemas de una red de datos de una gran planta en tiempo real, gran ancho de banda y baja latencia. Originalmente estos sistemas eran controles neumáticos que controlaban plantas pequeñas y evolucionaron en DCS.

Y finalmente, los SCADA que están ligados a las aplicaciones de distribución donde hay una necesidad de recolectar datos remotamente a través de redes potencialmente inseguras o con baja latencia y bajo ancho de banda. El SCADA emplea un control de lazo abierto sobre sitios separados geográficamente. Y usa terminales remotos para enviar datos de supervisión a un centro de control

3. Capítulo 3. Desarrollo

En este capítulo, el más extenso del documento, se recoge como se llevó a cabo el trabajo realizado para TAIM WESER para el proyecto de construcción de un SCADA Online. Se ha pretendido la realización de un SCADA Online que cumpla los objetivos presentados en el capítulo 1.

3.1 Descripción y esquemas del trabajo realizado

Se ha construido **una aplicación web en Python, desarrollada con los framework Django y Bootstrap, que se encuentra alojada dentro de un servidor Apache mediante la tecnología WSGI y que es capaz de hacer tareas de supervisión y adquisición de datos de cualquier marca de PLC mediante el estándar de interoperabilidad OPC**.

Antes de comenzar el proyecto había dos enfoques para construir el SCADA. Uno era realizar una aplicación de escritorio y el otro una aplicación web. La aplicación web, a grandes rasgos, es similar a una aplicación tradicional de escritorio pero es manejada a través de un navegador web.

Finalmente se optó por realizar una aplicación web. Hecho que otorga al SCADA más ventajas y permite alcanzar los objetivos con mayor facilidad. Esta decisión es la que moldeará en grandísima medida el aspecto final y la metodología de trabajo a seguir durante el proyecto. A continuación se recoge una lista de las ventajas de haber acabado optando por realizar una aplicación web, aunque en este documento se irá explicando con mayor detalle cuales son las características principales de una aplicación web:

- Las aplicaciones web funcionan a través de un navegador. Esto permite que el SCADA funcione en cualquier tipo de sistema operativo que tenga instalado un navegador web.
- El SCADA se podrá distribuir y actualizar de forma inmediata a todos los usuarios que lo necesiten. Ya que no se necesitan paquetes de instalación o actualizaciones

Para conseguir el objetivo de un control remoto por internet, que como se explicará más adelante no es otra cosa que manejar peticiones HTTP, se ha utilizado un servidor Apache por ser una solución de software libre.

Para el control de autómatas y la garantía de que el SCADA funcione con cualquier marca de PLC se ha utilizado el estándar OPC, que es un estándar para la interoperabilidad creado por la industria de la automática.

La configuración hardware del proyecto queda de la siguiente manera: el usuario accede al SCADA mediante un dispositivo que pueda ejecutar un navegador web, y todos los documentos y datos que el usuario necesite se generan en un único ordenador al que estarán conectados todos los PLCs. En este ordenador se ejecutarán las tareas pertinentes para devolver al usuario la información que solicite.

Es necesario recalcar que para este SCADA, al mismo ordenador al que se conecten los PLCs también deberá alojar la aplicación web y el servidor de internet. Así que si existen PLCs conectados a otro ordenador que no tenga la aplicación web y el servidor, no se podrán controlar mediante este SCADA.

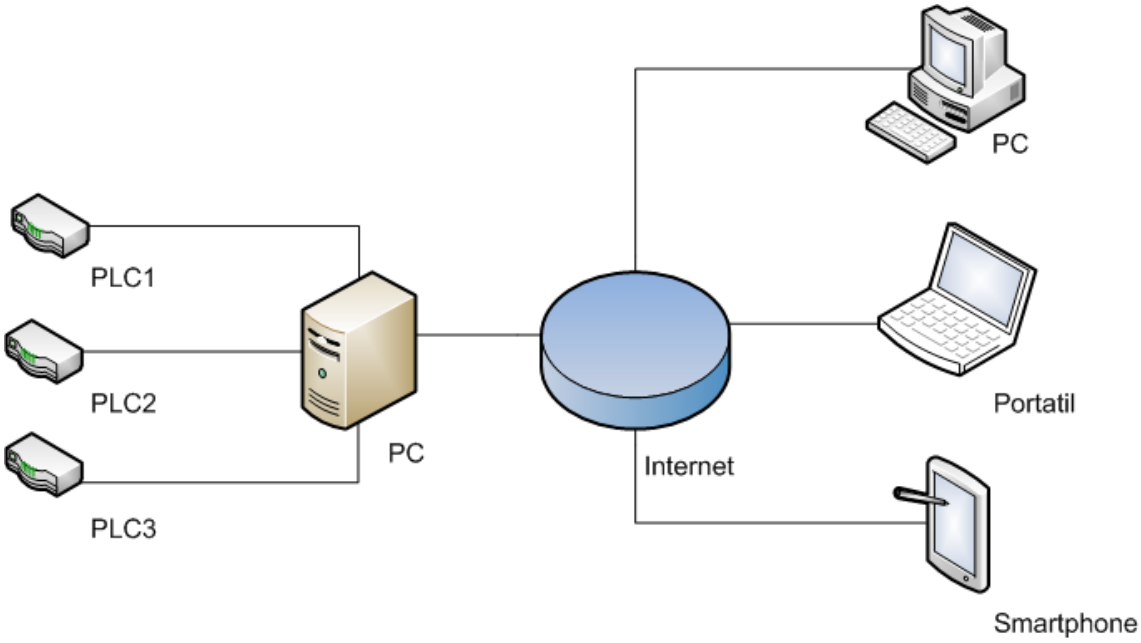


Figura 3, Esquema de conexión para el SCADA Online

Para explicar el apartado de software del proyecto se empleará todo lo que resta de este capítulo de la memoria. Y para fines didácticos se van a dividir todas las tareas que se realizan en el SCADA en bloques funcionales. En la siguiente figura se muestra un diagrama de bloques del proyecto a nivel de software:

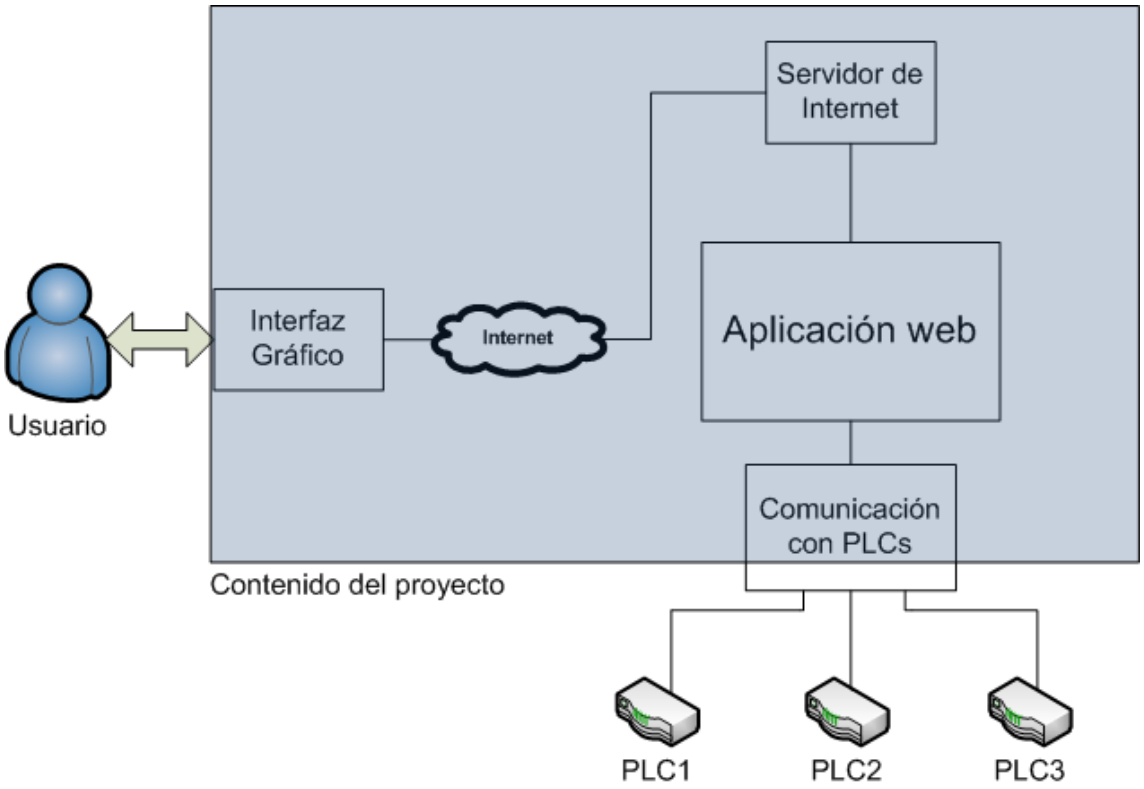


Figura 4 Diagrama de bloques de la plataforma software construida

Para mayor comprensión del lector, se explicarán a continuación de manera breve que es lo que hacen los distintos bloques:

1. Comunicación PLC: Es un conjunto de herramientas software que hace posible la comunicación con los distintos PLCs
2. Interfaz gráfico: Permitirá al usuario interactuar con los PLCs mediante iconos gráficos e indicadores visuales y mandará peticiones de información al servidor de internet.
3. Servidor de internet: Recibe las peticiones de información del usuario y envía las respuestas de la información solicitada permitiendo la comunicación vía internet. Gestiona peticiones y respuestas HTTP.
4. Aplicación web: La aplicación que se encargará de traducir las peticiones del usuario en órdenes para los PLCs y de generar los documentos para proveer al usuario con la información solicitada.

En los siguientes apartados de este capítulo se irá explicando con más detalle cada uno de los bloques, qué es lo que hace y que objetivos ayuda a cumplir. También cómo funciona y las tecnologías empleadas en este bloque, además de cómo interactúa con los demás bloques que le rodean.

3.2 Comunicación con los PLCs

Con este bloque se asegura la comunicación con PLCs, tanto lectura como escritura de variables e independientemente de la marca de los mismos. Y por lo tanto, en este apartado se hablará de como el protocolo de interoperabilidad OPC logra este objetivo.

La comunicación universal aporta grandes ventajas que no se tendrían si hubiera incompatibilidades con distintas marcas. Por ejemplo:

- A la hora de comprar nuevos PLCs, la marca sería un factor determinante que podría eliminar la posibilidad de adquirir PLCs de mayor calidad.
- No existe un gasto en tiempo y dinero que habría que invertir cada vez que se comprasen PLCs de otra marca para hacerlos compatibles con el sistema que ya hubiera.

La solución al problema radica en usar el estándar de interoperabilidad OPC creado por la OPC foundation para el propósito que aquí se quiere conseguir, la comunicación con un sistema de adquisición de datos con independencia del fabricante.

El protocolo OPC funciona con un esquema cliente-servidor. Esto significa que existen dos paquetes de software, uno para el servidor y otro para el cliente:

- Los PLCs conectados al ordenador se comunican con el servidor OPC, que transforma las instrucciones propias de la marca del PLC en instrucciones OPC.
- El servidor OPC se comunica con el cliente OPC, que transforma las instrucciones OPC en otras instrucciones de diverso tipo, ya sea python, C, java…

En la Figura 5 se ilustra lo explicado, empleando Python como lenguaje de las aplicaciones que harán uso de la comunicación con los PLCs. De este modo:

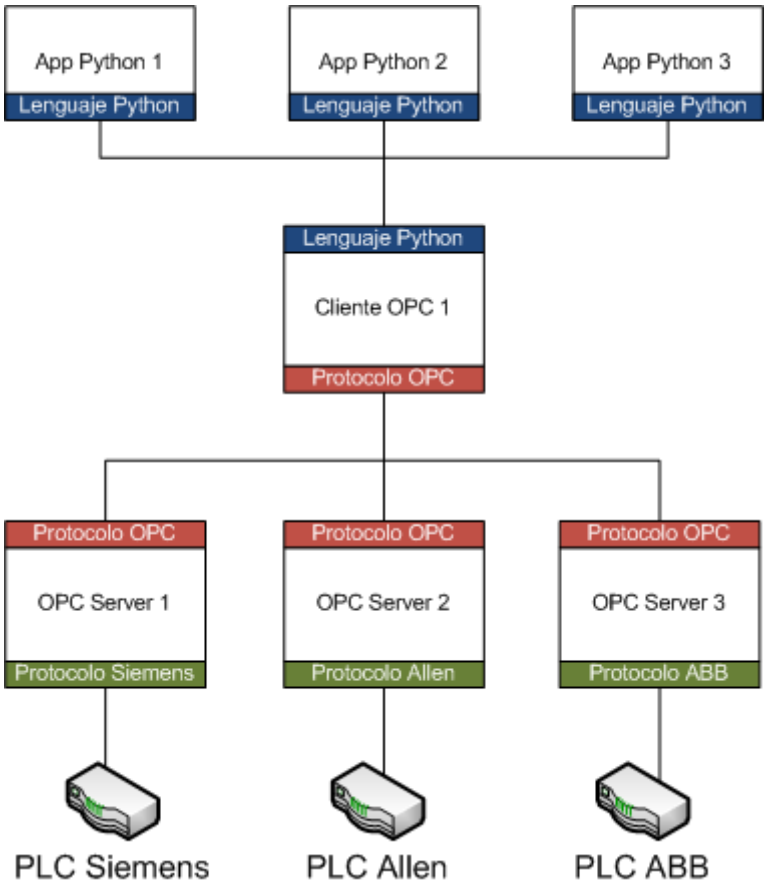


Figura 5 Diagrama Cliente-Servidor OPC

La Figura 5 es un ejemplo práctico de cómo funciona la tecnología OPC: los PLCs de distintas marcas se conectan a un servidor OPC que es capaz de comunicarse con el protocolo de cada marca, a su vez los servidores se comunican con el cliente OPC mediante el protocolo OPC y finalmente este cliente sirve los datos requeridos a las distintas aplicaciones Python.

3.2.1 Respecto a servidores OPC

Como ya se ha comentado, un servidor OPC traduce las instrucciones propias del PLC a instrucciones OPC. Por lo tanto, el servidor deberá conocer el protocolo de comunicación del PLC con el que se quiera comunicar. Los protocolos de los PLCs cambian de un fabricante a otro y son cerrados.

Así que para la creación de un servidor OPC se necesita acceso a una serie de herramientas de desarrollo de software no gratuitas que son los protocolos de cada marca de PLC. Esto hace que no se haya podido encontrar un servidor OPC de software libre, aunque existen servidores OPC desarrollados por las propias empresas que fabrican los PLCs o terceras compañías de software y actualmente, con la compra de PLCs se suelen distribuir también paquetes de software que incluyen servidores OPC , por lo que en principio resulta fácil acceder a estos.

Esto significa que **para poder usar el SCADA** creado en este proyecto **el usuario deberá adquirir** por su cuenta **un servidor OPC** que satisfaga sus necesidades.

Respecto a cómo funciona un servidor OPC y teniendo en cuenta que inicialmente uno se encuentra únicamente con una I/O analógica o digital:

- Cuando se programa la tarea a realizar por el PLC, se especifica que I/O digital o analógica será a su vez una variable OPC. Esto se hace en el propio compilador de la tarea
- A continuación, en el software que es el servidor OPC se añaden las variables del PLC que se deseen controlar. Así que ahora la I/O del PLC queda asignada a su servidor OPC correspondiente con un nombre específico.
- La I/O del PLC ha quedado etiquetada de manera univoca a una variable OPC.

Durante la realización de este proyecto se emplearon dos servidores OPC:

- Un servidor OPC con simulador de variables de PLCs para la fase de desarrollo del SCADA Online llamado “MatrikonOPC Server for Simulation”, de la empresa Matrikon.
- Y el segundo, un servidor de siemens llamado “S7-200 PC Access” que se usó para realizar pruebas sobre un PLC real, y en este caso de Siemens.

3.2.2 Respecto a los clientes OPC

El cliente OPC es una herramienta que se comunica con el servidor OPC para crear una aplicación, ya sea representar datos en gráficas, guardar registros en bases de datos o un listado de comandos en python para controlar PLCs, y este último caso es el de este proyecto, donde se ha escogido el cliente OPC “OpenOPC for Python” que será controlado por la aplicación web.

Este cliente OPC funciona mediante comandos en línea, lo que le otorga la ventaja especial de ser controlado mediante un script. En este script está el código para controlar los PLCs y la

aplicación web llamaría a este script para ejecutar las instrucciones pertinentes en función de lo que pida el usuario. Esto será explicado con mayor detalle en el apartado de la aplicación web.

Con el cliente "OpenOPC for Python" se pueden realizar las siguientes funciones:

- Adquisición de datos: Lectura y escritura de variable.
- Consulta de tiempo: A qué hora y fecha se ha realizado algo sobre la variable.
- Consulta de propiedades de la variable: como que tipo de variable es y si se puede escribir.

En la Figura 6, mediante consola Python se importa el cliente OPC llamado *OpenOPC*, y después de conectarse al servidor de *Matrikon* se efectúa la lectura y escritura de la variable *Random.Int4* y *Triangle Waves.Real8*.

```
>>> import OpenOPC
>>> opc = OpenOPC.client()
>>> opc.connect('Matrikon.OPC.Simulation')
>>> opc.read('Random.Int4')
(41, 'Good', '02/04/14 10:41:09')
>>> opc.write( ('Triangle Waves.Real8', 100.0) )
'Success'
>>> opc.properties( ['Random.Int2', 'Random.Int4', 'Random.String'], id=1)
[('Random.Int2', 'VT_I2'), ('Random.Int4', 'VT_I4'), ('Random.String', 'VT_BSTR'
)]
>>> opc.list('Simulation Items.Random.*Real*')
[u'Random.ArrayOfReal8', u'Random.Real4', u'Random.Real8']
>>>
```

Figura 6 Control por consola de comandos del cliente OPC "OpenOPC for Python"

La combinación de todas estas funciones permite alcanzar todos los objetivos requeridos en este proyecto en cuanto al tipo de tareas de adquisición de datos que se necesitan realizar sobre los PLCs.

3.3 Interfaz gráfico

El interfaz gráfico se ejecuta en el navegador web del usuario, y **es un documento HTML que usa javascript como lenguaje de programación.** A lo largo de este apartado se hablará de las distintas tecnologías empleadas para crear una página web dinámica, que es lo que se emplea como interfaz gráfico de usuario, además del framework Bootstrap para el desarrollo de los documentos HTML. También se hablará de: HTML, CSS, Javascript, JQuery, jqPlot y Bootstrap.

En las Figura 7 y Figura 8, se puede apreciar el interfaz gráfico sobre un navegador de escritorio y un navegador móvil respectivamente.

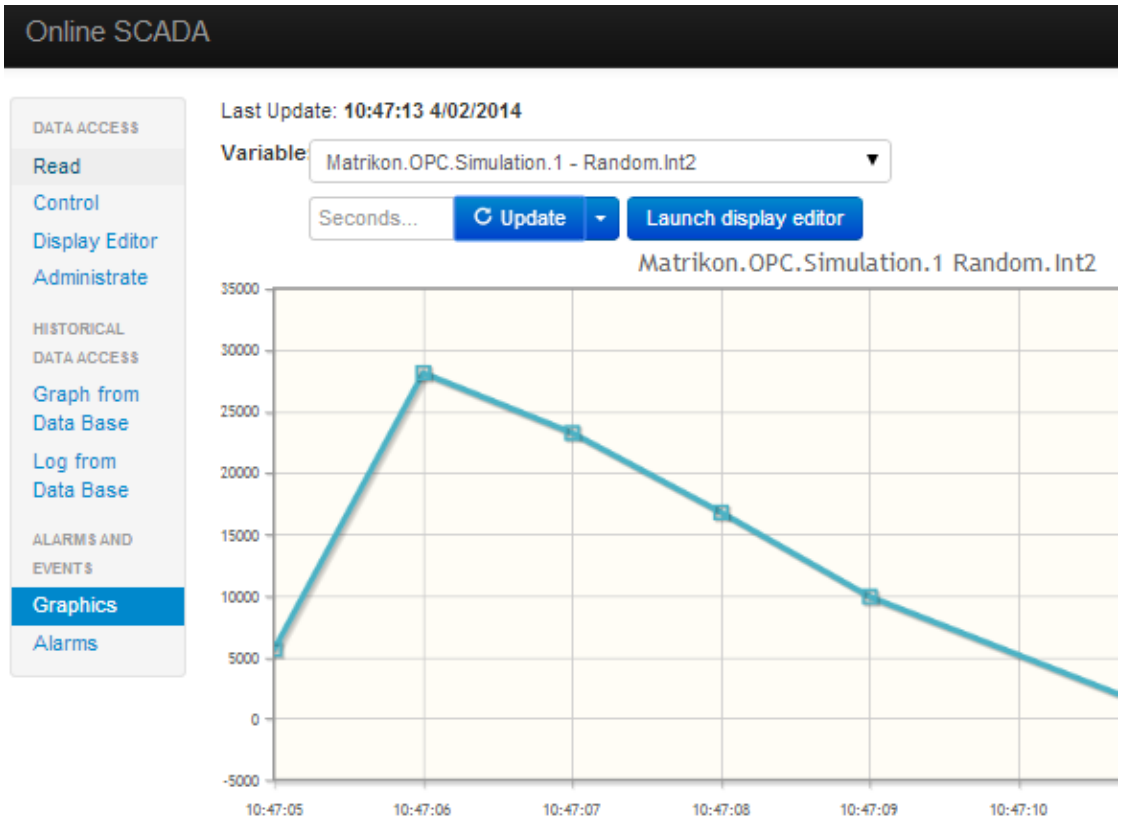


Figura 7 Interfaz gráfico en un navegador web de escritorio.

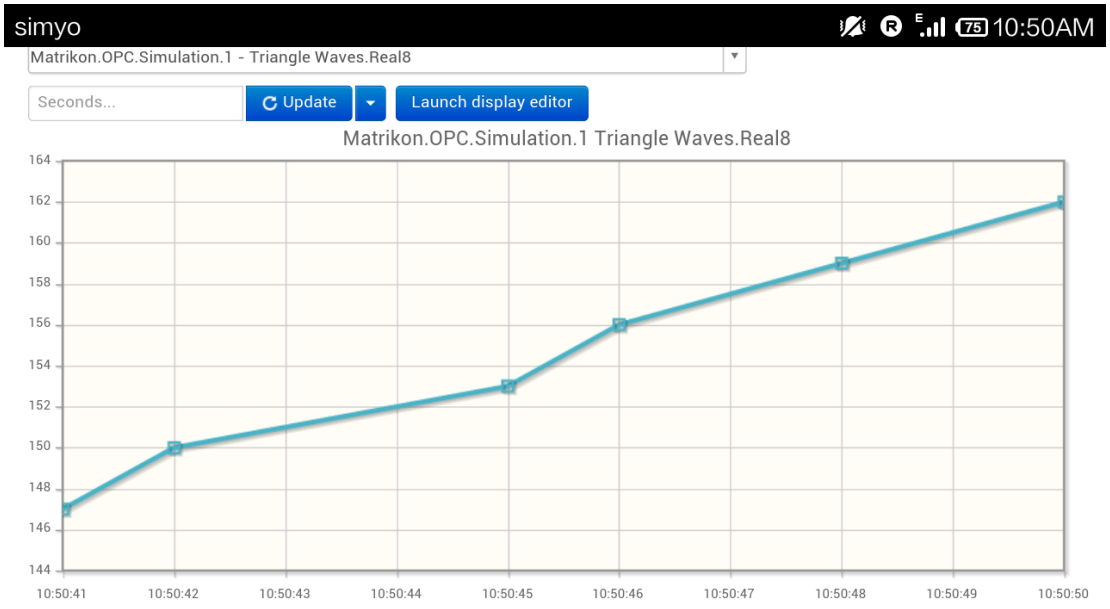


Figura 8 Interfaz gráfico en un dispositivo móvil

A continuación se explicarán los procesos involucrados en la creación del interfaz gráfico. Es decir, como llega el interfaz gráfico al navegador del usuario y que función desempeñan las distintas tecnologías empleadas en la creación del mismo.

Para que el usuario pueda acceder al interfaz gráfico para controlar el SCADA, ha de acceder a su navegador web e introducir una URL perteneciente al SCADA. Esta URL es una petición HTTP

al servidor que tiene alojada la aplicación web. Esto se explicará con más detalle en los próximos apartados. Entre la aplicación y el servidor se encargan de gestionar la petición enviada y devuelven, si todo ha ido bien, un fichero HTML que se ejecutará en el navegador del usuario.

3.3.1 Documentos HTML, CSS y Bootstrap.

El contenido en **HTML** del documento es el encargado de posicionar todos los elementos del documento, definiendo la estructura; tanto los párrafos con texto, los botones que desencadenan acciones y las tablas de información. Esto corresponde a la parte estática del interfaz gráfico, pues ese documento HTML permanece de forma invariante en el servidor y se envía cada vez que es solicitado. Además, el documento HTML contiene todo el código de las funciones Javascript que se necesitan para permitir la funcionalidad del interfaz.

```
152
153
154             <!-- div class="span12" id="tabla"-->
155             <table id="tabla" class="table table-striped table-bordered table-condensed">
156               <thead>
157                 <tr>
158                   <th style="width: 30%">Server</th>
159                   <th style="width: 30%">Name</th>
160                   <th style="width: 40%">Value</th>
```

Figura 9 Extracto de un fichero con contenido HTML.

Por otra parte, se usa **CSS** como lenguaje de estilo para dar aspecto y formato al contenido mostrado mediante HTML. Se usa para dar una buena presentación al documento, ganando usabilidad. Todo lo referente a CSS, que es lo mismo que hablar de la estética, en este proyecto viene dado por las librerías del framework Bootstrap. El resultado final es una página web en la que además se ha buscado un diseño gráfico bonito, empleando por ejemplo botones y marcos redondeados de distintos colores, como se aprecia en la Figura 10.

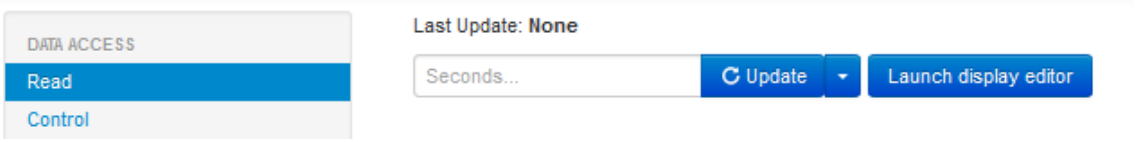


Figura 10, Formato dado a la página web.

La importancia de usar **Bootstrap** como framework de trabajo para el desarrollo de los documentos HTML y CSS ha sido fundamental, pues dota de un diseño gráfico que se ha convertido en un referente actual y es similar al empleado en la página web Twitter y que además permite un uso en dispositivos móviles. Esto puede parecer algo trivial en un principio, pero representar la información de forma elegante e intuitiva al usuario es algo que busca cualquier desarrollador de páginas web pues permite que la experiencia de manejar el SCADA sea algo agradable y rápido. De esta manera se puede conseguir que el cliente utilice una solución SCADA frente a otra solo por el hecho de que resulte más atractivo de usar.

3.3.2 Lenguaje de programación Javascript, jqPlot y librerías jQuery

Javascript es el lenguaje de programación usado en páginas web. Se trata de código que es alojado en la parte del usuario, dentro del fichero HTML, y se emplean para interactuar con el usuario, controlar el navegador, comunicarse de forma asíncrona con la aplicación web y

alterar el contenido del documento a mostrar. Un ejemplo de cómo funciona una función javascript que realiza las funciones descritas aparece en la Figura 11:

Figura 11 Función Javascript con comunicación asíncrona y manipulación de documento HTML.

De esta forma se ha obtenido información de forma asíncrona que se ha alojado en el mismo documento HTML original. El hecho de que no se solicite un nuevo documento HTML, sino que sea el documento original el que vaya cambiando es lo que dota de dinamismo al interfaz gráfico. Y esto, que aunque no se pueda apreciar de forma tan directa como es diseño del interfaz, es otra de las características que definen a este interfaz gráfico. Es decir, un **interfaz dinámico**.

Hasta ahora se ha hablado exclusivamente de Javascript como un lenguaje de programación aislado. Pero también hay que tener en cuenta la librería de funciones JavaScript que se ha estado usando, jQuery. Como cualquier librería, sirve para implementar funciones de forma más rápida y en concreto, está orientada a facilitar la manipulación del documento HTML, manejo de eventos y comunicación asíncrona. En el Anexo 6.2 se puede ver ejemplos del uso de estas librerías.

Como combinación del lenguaje de programación Javascript y la librería jQuery se desarrolló *jqPlot*, que es un plugin para realizar gráficas dentro de documentos HTML y el autor ha empleado en este proyecto para visualización de variables tanto en tiempo real como para representar información de bases de datos.

En el Anexo 6.2 se presentará el código con detalle y aquí se explicará la funcionalidad del plugin *jqPlot*. Su misión es representar datos en un elemento del documento HTML. Los datos, en este caso de los PLCs, son suministrados al plugin *jqPlot* para que este cree el elemento del documento HTML que el usuario ve como una gráfica. En la Figura 12 se observa una gráfica creada mediante el plugin *jqPlot*.

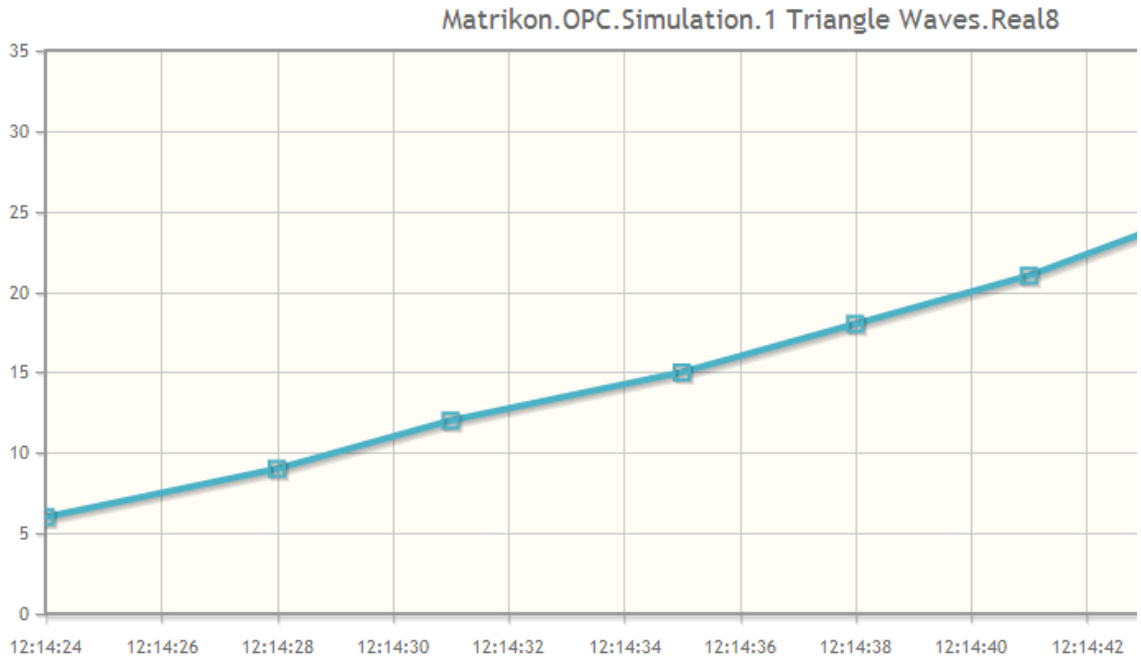


Figura 12 Gráfica construida mediante el plugin jqPlot

3.4 El servidor de internet

Gracias al servidor de internet se tiene la posibilidad de una comunicación remota, para alcanzar el control de los PLCs mediante internet. Dicho de otra manera, el servidor de internet gestiona todas las peticiones y respuestas HTTP, y se queda como **el nexo de unión entre el interfaz gráfico y la aplicación web Python.** Sirve pues para enviar al usuario todos los ficheros y datos que requiera para hacer uso del SCADA. En la siguiente figura se ha representado con un diagrama de bloques el servidor de Internet:

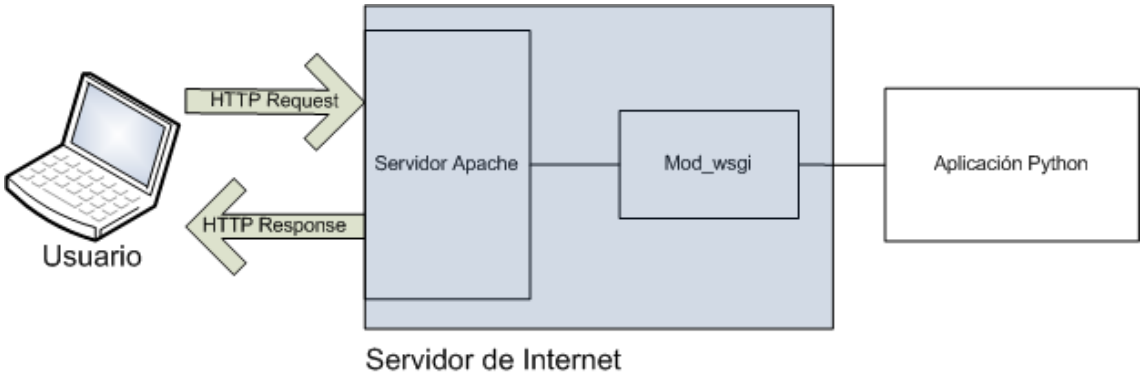


Figura 13 Diagrama de bloques del servidor de Internet

3.4.1 Diálogo HTTP

A continuación se detallará en qué consiste el concepto de peticiones y respuestas HTTP del que se ha hablado anteriormente, y como se usan estos elementos para alcanzar los objetivos del proyecto.

Un servidor lo que hace es recibir peticiones HTTP del usuario y manda de vuelta respuestas HTTP. Las peticiones HTTP son solicitudes de información por parte del usuario y las respuestas HTTP son los datos de respuesta del servidor ante la solicitud. La petición HTTP del usuario es una llamada a una url dentro del dominio del servidor web y se puede realizar de forma asíncrona, esto es cada vez que el usuario necesite información puede solicitarla inmediatamente. Por otra parte, las respuestas HTTP del servidor pueden ser estáticas, cuando los datos de la respuesta son ficheros alojados en el servidor; o dinámica, cuando los datos de la respuesta han sido generados por la aplicación web a partir del momento que el usuario ha mandado la petición.

Para realizar peticiones HTTP en este proyecto se han usado los métodos GET y POST, cómo se puede ver en los anexos. A continuación se explican en qué consisten estos métodos:

- **GET**: Se solicita cierta información al servidor. Este método solo debería devolver información, el usuario no mandaría datos. Por ejemplo cuando se entra a una url determinada, el servidor devuelve un documento HTML. O cuando el usuario solicita actualizar las variables, el servidor devolverá los nuevos valores. Pero en ningún caso el usuario manda datos al servidor.

  La siguiente imagen es la respuesta del servidor ante una petición HTTP de tipo GET. La respuesta, en formato JSON, contiene el valor actualizado de las variables contenidas en el servidor OPC.

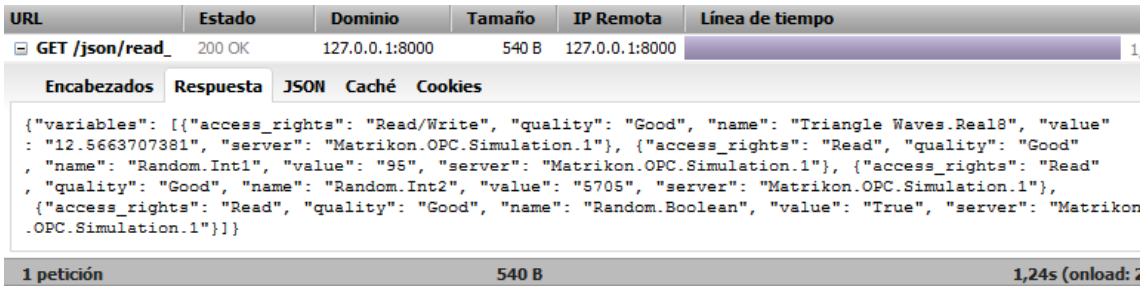


Figura 14 Respuesta del servidor ante una petición HTTP de tipo GET.

- **POST**: Similar al GET, con la diferencia de que aquí el usuario manda información al servidor. Por ejemplo si el usuario quiere cambiar el valor de algunas variables, se mandará al servidor el nombre de las variables y sus nuevos valores para que ejecute las operaciones. En la siguiente imagen se observa los datos enviados por el usuario mediante una petición HTTP tipo POST al servidor de internet. Los datos están en formato JSON y contienen el valor a escribir sobre una serie de variables. La respuesta del servidor ante este tipo de petición es igual que ante una petición GET.

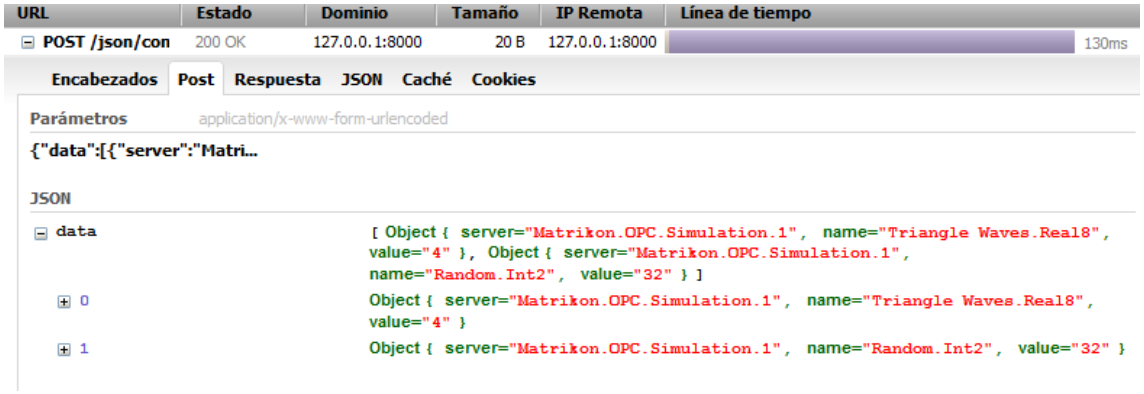


Figura 15 Petición HTTP con método POST.

En la siguiente figura se observa la respuesta HTTP del servidor. Consiste en una string en formato JSON donde se informa del resultado de ciertos sucesos.

Figura 16 Respuesta HTTP del servidor ante una petición HTTP tipo POST.

Como se puede observar, cuando se intercambia información entre el servidor y el cliente se usa el estándar abierto JSON. Esto es porque para poder transmitir la información ha de hacerse en un formato específico y por lo tanto no se pueden enviar las variables que usan Javascript o la aplicación Python directamente, han de transformarse primero al formato JSON para ser después enviadas.

En definitiva, JSON transforma cualquier tipo de formato, por ejemplo Javascript o Python, en una string para que se pueda transmitir mediante el protocolo HTTP. Y por lo tanto, existirán funciones, tanto en Javascript y Python, para codificar y descodificar elementos JSON.

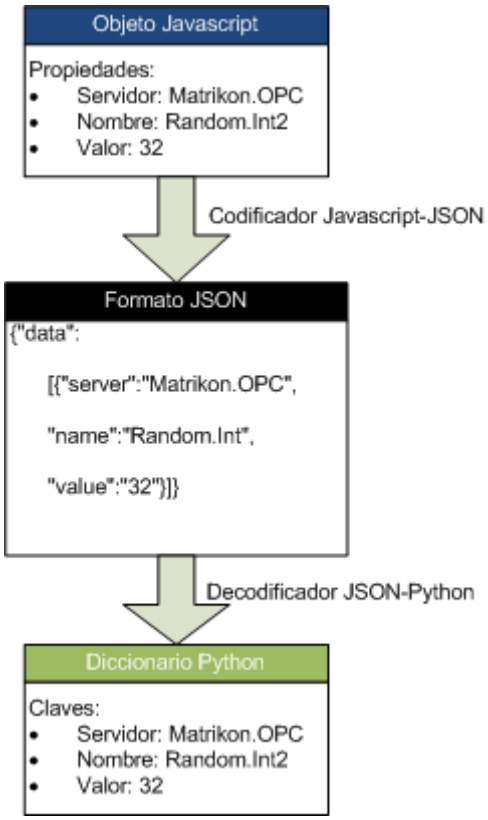


Figura 17 Diagrama de flujo para la conversión de diversos formatos en JSON

3.4.2 Componentes del servidor de Internet

Como se ha expuesto en el diagrama de bloques al principio de este apartado, el servidor de internet está compuesto de un servidor Apache y un módulo Apache llamado “mod_wsgi”.

Mientras que el servidor Apache se encarga de gestionar todas las peticiones y respuestas HTTP, el módulo Apache “mod_wsgi” sirve para alojar la aplicación Python dentro del servidor Apache. Este módulo usa la especificación “wsgi” (Web Server Gateway interface) que establece como se comunican las aplicaciones web con un servidor web, y más concretamente “mod_wsgi” permite la comunicación con aplicaciones python.

En la siguiente figura se muestra cómo se ha modificado la configuración de Apache para incluir el modulo “mod_wsgi”:

- Con la directiva “WSGIScriptAlias” el usuario indica que cuando se acceda a una determinada url ha de destinarse esa petición HTTP a la aplicación Python.
- Las demás directivas son para importar distintos módulos python. Tiene que ver con configuración relativa al funcionamiento interno de Python.

```
WSGIScriptAlias / C:/Users/Ricardo/Dropbox/django_scada/django_scada/apache/wsgi.py
WSGIPythonPath C:/Users/Ricardo/Dropbox/django_scada/
WSGIPythonPath C:/Users/Ricardo/Dropbox/django_scada:C:/Python27/Lib/site-packages
Alias /static/ C:/Users/Ricardo/Dropbox/static/
```

Figura 18 Configuración del servidor Apache para alojar la aplicación Python.

3.4.3 Seguridad

En este apartado se discutirá la seguridad que proporciona el sistema ante un usuario que desease cambiar o acceder a los ficheros de código fuente o cualquier otro tipo de información a la que no debería poder acceder.

En un principio, se puede decir que la plataforma SCADA entraña un compromiso de seguridad importante ya que permite controlar PLCs de forma remota. Pero teniendo en cuenta esa característica intrínseca, ha de averiguarse cuales son las posibilidades de realizar cambios para que el SCADA se comporte de forma no debida.

El servidor Apache permite al usuario acceder a un directorio local, respecto del ordenador al que están conectados los PLCs, donde habría un conjunto de archivos que han de ser servidos al usuario. Estos archivos serían documentos HTML, librerías de funciones Javascript y CSS e imágenes. En los documentos de Apache se especifica que ese directorio en ningún momento ha de incluir ficheros de configuración del servidor Apache o una aplicación web, ni tampoco el código fuente de una aplicación web. Respetando esta directiva, el servidor Apache nunca suministrará archivos comprometidos.

Este tipo de archivos locales a los que se puede se acceder se les da el nombre de *ficheros locales*, y como ya se ha indicado nunca han de comprometer la seguridad del sistema.

Otro tema de vulnerabilidad sería la facilidad para acceder al SCADA sin los credenciales requeridos. Este tema asociado con la criptografía está resuelto por métodos de protección estándar en páginas web y por tanto no existe un riesgo específico por la dificultad de adivinar o acceder al sistema mediante credenciales falsos. Las funciones para acreditar el acceso de un usuario al SCADA vienen dadas por el framework de aplicaciones web Django.

3.5 La aplicación web

Cómo se ha explicado, el servidor de internet da respuestas HTTP ante peticiones HTTP del usuario. La aplicación web es la encargada de proporcionar las respuestas estáticas y dinámicas del servidor. Con respuestas dinámicas se entiende un tipo de información que ha de ser generada sobre la marcha y por lo tanto no estaba almacenada previamente en el servidor. La aplicación web es ejecutada en el mismo ordenador que tiene el servidor de internet, es capaz de comunicarse en HTTP y genera el contenido que el navegador necesita.

A lo largo de este proceso se hablará de la aplicación web, cuyo contenido ha sido desarrollado exclusivamente por el autor, y es el componente principal de este proyecto. A continuación se hablará de cómo el framework Django da forma estructural a la aplicación web, los modelos creados para la representación de datos en la aplicación y las distintas tareas que realiza, ya sea supervisión y control de datos y actividades periódicas.

Como simplificación general se puede decir que la aplicación web genera la información necesaria para construir el SCADA y la cadena de procesos para generar el contenido estático o dinámico de la aplicación web es el siguiente:

1. El usuario desea hacer algo y realiza una acción que desencadena un evento. Ejemplo: apretar un botón de actualización.
2. Ese evento es una función de llamada a una url determinada. Es decir, se envía una petición HTTP al servidor.
3. El servidor Apache recibe la petición y traslada la orden a la aplicación web python mediante “mod_wsgi”.
4. La aplicación web recibe la instrucción y ejecuta la función asociada a la url a la que se ha llamado inicialmente.
5. La función de la aplicación web genera una salida que es transmitida al servidor Apache para que este la transforme en una respuesta HTTP, y que recibirá el navegador web del usuario.
6. Finalmente, el navegador web procesará la información recibida. Por ejemplo, actualizando los valores de las variables que se vean por pantalla.

Para llevar a cabo el paso cuatro que se ha descrito arriba, en la siguiente imagen se ilustra el fichero Python que asocia una url con una determinada función. En este caso, la url “/home” tiene la función asociada “variables.views.home”, una función que devuelve una plantilla html:

```
from django.conf import settings


admin.autodiscover()

urlpatterns = patterns('',
    #url(r'^test/', 'variables.views.test'),
    #url(r'^test_json/', 'variables.views.test_json'),

    url(r'^home/', 'variables.views.home'),
    url(r'^read/', 'variables.views.read'),
    url(r'^json/read_update/','variables.views.read_update'),
```

Figura 19 Esquema de urls con sus funciones asociadas de un fichero Python

3.5.1 Django como framework de la aplicación web

Django ha sido el framework empleado para construir la aplicación web y emplea el lenguaje de programación Python. Algunas de las ventajas del uso de Django son las siguientes:

- Está orientado a crear aplicaciones web rápidamente. Pues incorpora, es decir, no ha de emplearse tiempo en escribirlas; funciones generales como: registro de usuarios, gestión de peticiones y respuestas HTTP.
- Django se ha diseñado para hacer aplicaciones web escalables. Es decir, una vez se haya construido una aplicación se pueden acoplar más funcionalidades de manera sencilla.
- Existe una comunidad de código abierto como equipo de desarrollo de Django. Y por lo tanto, resulta sencillo encontrar documentos o foros donde poder consultar dudas. Por lo que se pueden crear utilidades de forma rápida.

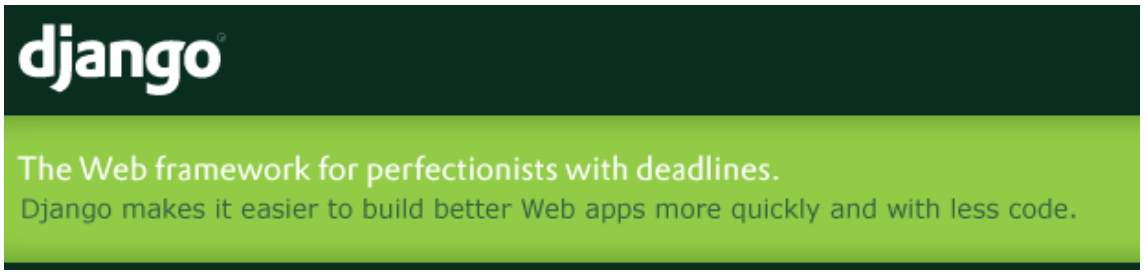


Figura 20 Logo de Django, framework usado para construir la aplicación web.

Django establece como punto de partida unos ficheros configurables y expandibles que serán los bloques fundacionales de la aplicación que se desea construir para alcanzar los objetivos deseados.

La siguiente imagen es del directorio de la aplicación que se ha creado para este SCADA:

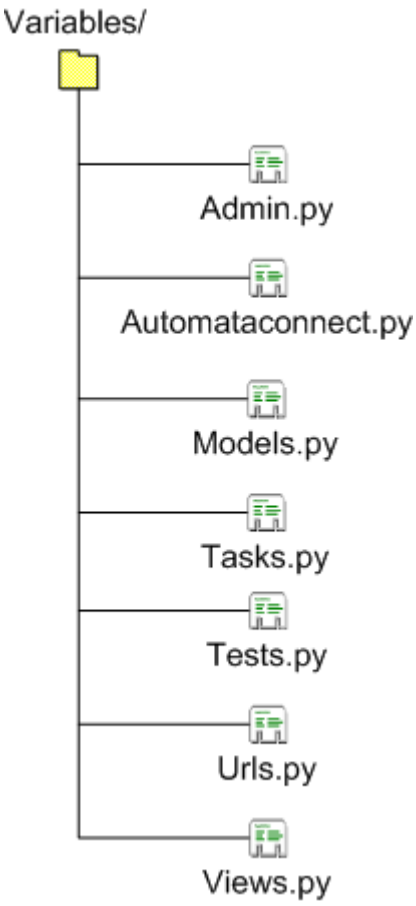


Figura 21 Estructura del directorio que contiene la aplicación creada para este proyecto.

La carpeta "Variables" contiene los ficheros python, con extensión ".py", y su contenido es creado en su totalidad, obviando ciertas directivas estándar, por el autor para dar forma al proyecto. Cada fichero python está encaminado a cumplir una serie de tareas que serán explicadas brevemente en el siguiente párrafo. De esta forma:

- Admin.py: Se encarga de tareas administrativas. Tales como añadir datos a la base de datos.
- Automataconnect.py: Es un script realizado por el autor y sirve para comunicar la aplicación web con el cliente OPC.
- Models.py: Contiene la información acerca de las variables especiales creadas para esta aplicación web. El nombre que reciben este tipo de variables especiales es Modelos.
- Tasks.py: Aquí se encuentran unas funciones especiales que son las tareas programables del proyecto. Con programables se quiere decir que se pueden ejecutar cíclicamente cada cierto periodo de tiempo. Como por ejemplo añadir el estado de una variable a la base de datos cada cierto intervalo de tiempo.
- Tests.py: Se trata de un fichero usado en la fase de desarrollo para tareas de testeo. Y es irrelevante en la aplicación final.
- Urls.py: Es el fichero que asocia cada url con una función determinada.

- Views.py: Todo el resto de funciones para que la aplicación web pueda funcionar se encuentran aquí. Como consultar el estado de una variable por ejemplo. Aquí se encuentra prácticamente todo el código realizado para la aplicación web.

3.5.2 Los modelos de la aplicación web

Con Django, uno puede crear variables personalizadas para la aplicación que desea crear. Esto se hace a través de *modelos*. Un *modelo* es una manera de almacenar información pues puede contener los campos y propiedades que se deseen. Visto desde el punto de vista de la base de datos un *modelo* es una tabla, como las de una hoja de cálculo, y cada campo del *modelo* correspondería a un campo de la tabla. Pero siendo más técnico un *modelo* se asocia al concepto de *clase* en la programación orientada a objetos.

A lo largo de este apartado se tratará sobre los *modelos* que se han definido para esta aplicación, que propiedades contienen y como una vez creados el usuario introduce información nueva.

Para esta aplicación web se han creado dos *modelos*. Uno para poder representar variables de los PLCs y el otro para guardar un registro histórico de los valores de los PLCs. El código que los define se encuentra en el fichero *models.py*.

El modelo *Variable* es empleado para la representación de variables del PLC y contiene una serie de campos para representar propiedades relacionadas con una variable OPC. Así se ve en la siguiente figura:

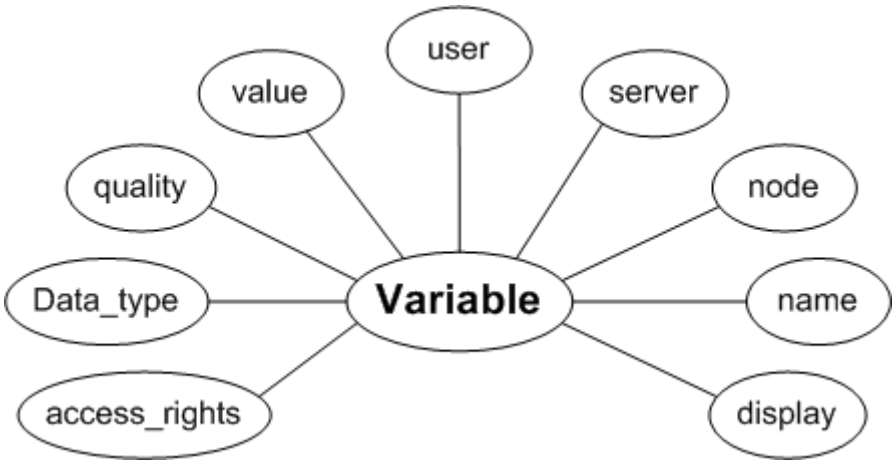


Figura 22 Modelo Variable y sus distintas propiedades.

A continuación se explicará la figura superior:

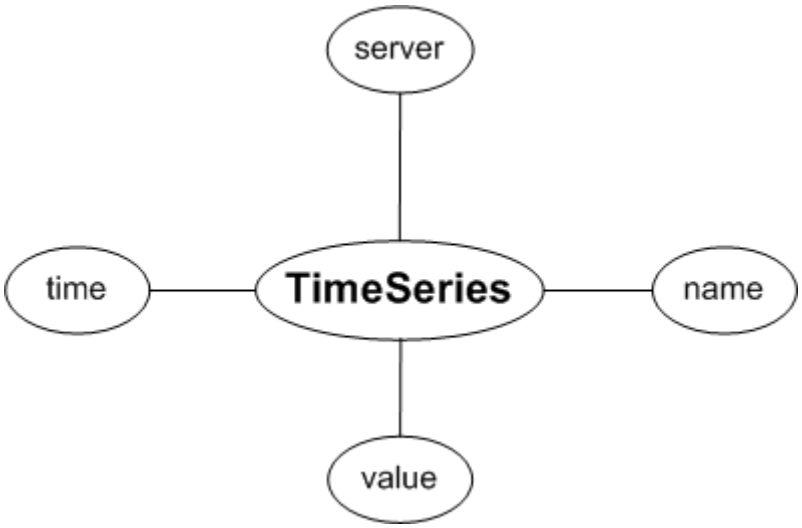
- Variable: Es la *clase* que se usa para crear objetos de tipo *Variable*
  - User: Hace referencia al usuario que controla la variable. El usuario a su vez también es un tipo de *modelo* que trae Django creado en sus archivos fuente y por lo tanto no tiene que ser configurado.
  - Server: Con este campo se asocia el servidor OPC de la variable.
  - Node: Dentro de un servidor, en qué nodo se encuentra.
  - Name: El nombre de la variable.
  - Value: Su valor en un momento dado.

- Quality: Si la variable se ha podido leer correctamente, o no, se indicará con este campo.
- Data_type: Esto es una herencia del tipo de variables OPC. Las variables OPC pueden ser, por ejemplo, enteros de 4 bytes con signo y esto ha de ser indicado.
- Access_rights: Aquí se especifica si la variable es de escritura o lectura.
- Display: Si el usuario desea que sea visible en las distintas operaciones de control del SCADA.

Poniendo un ejemplo, lo explicado quedaría así en una base de datos:

Base de datos								
User	**Server**	**Node**	**Name**	**Value**	**Quality**	**Data_type**	**Access_rights**	**Display**
Ricardo	Matrikon	Random	Int2	32	good	VT_4	write	True
Antonio	SiemensOPC	None	Real4	45	good	VT_4	read	True

El otro *modelo*, llamado *TimeSeries,* es para el registro histórico del estado de las variables de los PLCs. Esto es porque se querrá hacer una visualización histórica del estado de alguna de las variables. El modelo queda representado gráficamente de la siguiente manera:



- TimeSeries: Es la *clase* que se usará para crear objetos de tipo *TimeSeries*.
  - Server: Nombre del servidor donde está alojada la variable.
  - Name: El nombre de la variable.
  - Value: El valor de la variable en un momento dado.
  - Time: Registro de la fecha y hora de lectura de la variable.

Una vez explicado el proceso de configuración de los *modelos* se va a hacer hincapié en la creación de los mismos.

Cada vez que se quiera incluir un nuevo modelo, por ejemplo una nueva variable a controlar de un PLC, lo que se hace es introducir en la base de datos la nueva variable para que el usuario pueda acceder a ella cuando lo necesite.

Para poder realizar esto, la aplicación web ha de crear una página en el SCADA donde el usuario pueda a través de un formulario introducir los nuevos campos de la variable que desea controlar y cuando la aplicación web reciba la solicitud, almacene la información en la base de datos. Todas estas tareas se crean automáticamente con el framework Django. Y únicamente el autor ha de configurar el archivo *admin.py* para configurar que mostrará el formulario que rellene el usuario, como se ve a continuación:

```
class VariableAdmin(admin.ModelAdmin):
    fieldsets = [
        ('OPC Server',              {'fields': ['server']}),
        ('Variable Name', {'fields': ['name']}),
    ]

    list_display = ('name','server')

    search_fields = ['server','name']
```

Figura 23 Código de *admins.py* para la configuración del formulario para incluir nuevas variables.

Este código indica cuales son los campos que ha de rellenar en el formulario el usuario para poder incluir una nueva variable. Se puede ver que solo se requieren los campos de servidor y nombre de la variable, los demás campos se rellenaran posteriormente con otras funciones. Django con esta información y la de sus ficheros fuente crea el siguiente formulario que es accesible por el usuario:

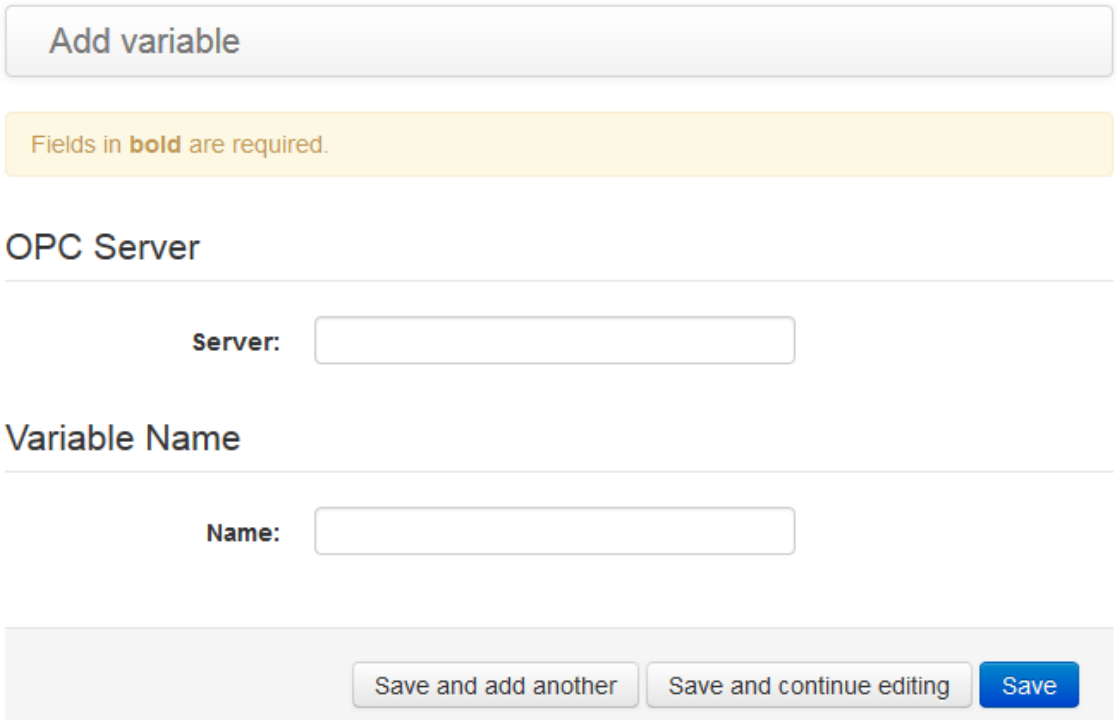


Figura 24 Formulario para la creación de una nueva variable a controlar en la base de datos

Hay que tener en cuenta que las variables que un usuario guarde en su cuenta, no se asignarán a la cuenta de otro usuario. De esta manera, cada usuario tendrá un SCADA con las variables a controlar que el desee. Es decir, cada usuario tendrá un SCADA personalizado a sus necesidades.

Para el otro tipo de *modelo* para el registro histórico de variables el proceso es algo diferente al tratarse de tareas programables en el tiempo y cómo se crean nuevas tareas será tratado en el apartado *3.5.4* de este capítulo.

Finalmente, cuando el usuario desee guardar información sobre los modelos se guardarán en una base de datos que viene incluida en las librerías que forman Python llamada SQLite.

3.5.3 Supervisión y adquisición de datos.

El SCADA, y por lo tanto la aplicación web, ha de proveer al usuario con tareas de supervisión y adquisición de datos. Este tipo de tareas son la lectura y escritura de variables de los PLCs. Este tipo de tareas recibe y envía datos al usuario mediante el protocolo HTTP y se comunica con los PLCs mediante el cliente OPC. Cómo se reciben y envían esos datos ya se ha tratado previamente así que este apartado se centra en la comunicación de la aplicación web con el cliente OPC y los algoritmos para las funciones de supervisión y adquisición de datos.

La aplicación web para comunicarse con el cliente OPC utiliza un script realizado por el autor llamado a*utomataconnect.py.* A este script se le importa el cliente OPC en forma de módulo python que recibe el nombre de *OpenOPC.* El módulo python es una librería de funciones del cliente OPC. El script además tiene dos funciones python: una para leer y otra para escribir sobre variables de PLCs. Como argumentos se envían nombre y servidor en el primer caso; y nombre, servidor y valor de escritura en el segundo.

```
from variables import OpenOPC

def read(server,name):
    try:
        opc = OpenOPC.open_client('localhost')
        opc.connect(server)
        value, quality, time = opc.read(name)
        access_rights=opc.properties(name, id=5)
        opc.close()
        value=str(value)
        return value,quality,access_rights,time

    except Exception:
        opc.close()
        return None,'Error',None,None
```

Figura 25 Código del script *automataconnect.py* para lectura de variables.

El usuario accede a las tareas de supervisión y adquisición de datos en el interfaz desde tres páginas web distintas: una con una tabla con los valores de las variables, otra con una gráfica en tiempo real del estado de una variable y la última también con una tabla pero para la escritura de valores. Estas tres páginas web utilizan el navegador web para crear la interfaz pero necesitan de datos y documentos que provee la aplicación web.

En una primera instancia la aplicación web recibe una petición HTTP cuando el usuario accede a una de estas páginas, y la aplicación devuelve en una respuesta HTTP el documento HTML correspondiente a la página web solicitada y un diccionario python. Este diccionario Python contiene los valores e información acerca de todas las variables controladas por el usuario y la sintaxis del documento HTML se encarga de colocar esta información en el interfaz de usuario. Con llamadas al script *automataconnect.py* se obtienen los datos que alberga el diccionario python .

```
@login_required
def read(request):
    variable_list=get_variables(request.user)
    return render_to_response('read.html', {'variable_list': variable_list} ,
                              context_instance=RequestContext(request))

@login_required
def control(request):
    variable_list=get_writable_variables(request.user)
    return render_to_response('control.html', {'variable_list': variable_list} ,
                              context_instance=RequestContext(request))
```

Figura 26 Código para el envío de respuestas HTTP con un documento HTML y un diccionario python.

Una vez que el usuario tiene en su navegador web la página web solicitada, este puede querer actualizar los valores de la variable o enviar a que valores desea escribir en ciertas variables. De esta manera, se realiza una comunicación asíncrona navegador-servidor para que la aplicación web ejecute la función correspondiente. Nótese que en ninguno de estos casos se vuelve a enviar un documento HTML, sino que se envían datos en formato JSON creados en el mismo instante que los solicita el usuario. Se trata únicamente de respuestas dinámicas. En esta segunda instancia se emplea el siguiente flujo de operaciones para las páginas web descritas:

- Para actualización de tabla de valores: Cuando el usuario manda la petición HTTP para realizar esta operación, la función python llama al script *automataconnect.py* para obtener de nuevo un diccionario con la información completa de las variables de los PLCs. El diccionario se codifica en formato JSON.
- Para actualizar la gráfica de una variable: El usuario manda la variable que desea actualizar y la función python devuelve una string en formato JSON dando la información de la variable y el instante de tiempo en que fue capturada .
- Para escritura de variables: Se decodifica del formato JSON los nuevos valores de las variables que el usuario ha enviado, y se escriben en los PLCs llamando al script *automaconnect.py* para que finalmente se devuelva al usuario, en formato JSON, mensajes informando si se ha podido o no llevar a cabo la operación.

Mediante este apartado se ha pretendido dar a conocer conceptualmente los algoritmos de control pero para más detalle en profundidad acerca de las funciones y código python del fichero *views.py* se puede consultar el Anexo 6.1.4.

3.5.4 Actividades periódicas: Almacenaje en base de datos.

Una característica interesante del SCADA es la posibilidad de incluir un registro histórico de las variables de PLCs para que de esta manera se puedan representar gráficas temporales de la

evolución de una variable a lo largo del tiempo. Para lograr esto es necesario estar almacenando durante un periodo de tiempo muestras de la variable a una frecuencia determinada. Algo que rítmicamente guarde información en la base de datos. Esto se consigue mediante un ejecutor de código acorde a un horario llamado Celery.

El usuario deberá configurar qué variable quiere controlar y con qué frecuencia mediante el interfaz gráfico del navegador web. Configurando los ficheros de Django y Celery se le muestra el siguiente formulario web al usuario. Donde se puede ver que aparecen los campos necesarios en el siguiente orden de aparición:

- Nombre que se le da a esta tarea periódica: “Muestro de variable Triangle Wave4”
- El nombre de la función que ha de llevarse a cabo periódicamente: “variables.tasks.store_value”, que es la función encargada de almecenar el valor de una variable en la base de datos.
- El intervalo de tiempo, que son cinco minutos en este ejemplo.
- Los argumentos de la variable escritos en forma de string, que son el servidor “Matrikon.OPC.Simulation1” y el nombre de la variable “Triangle Waves.Real8”.

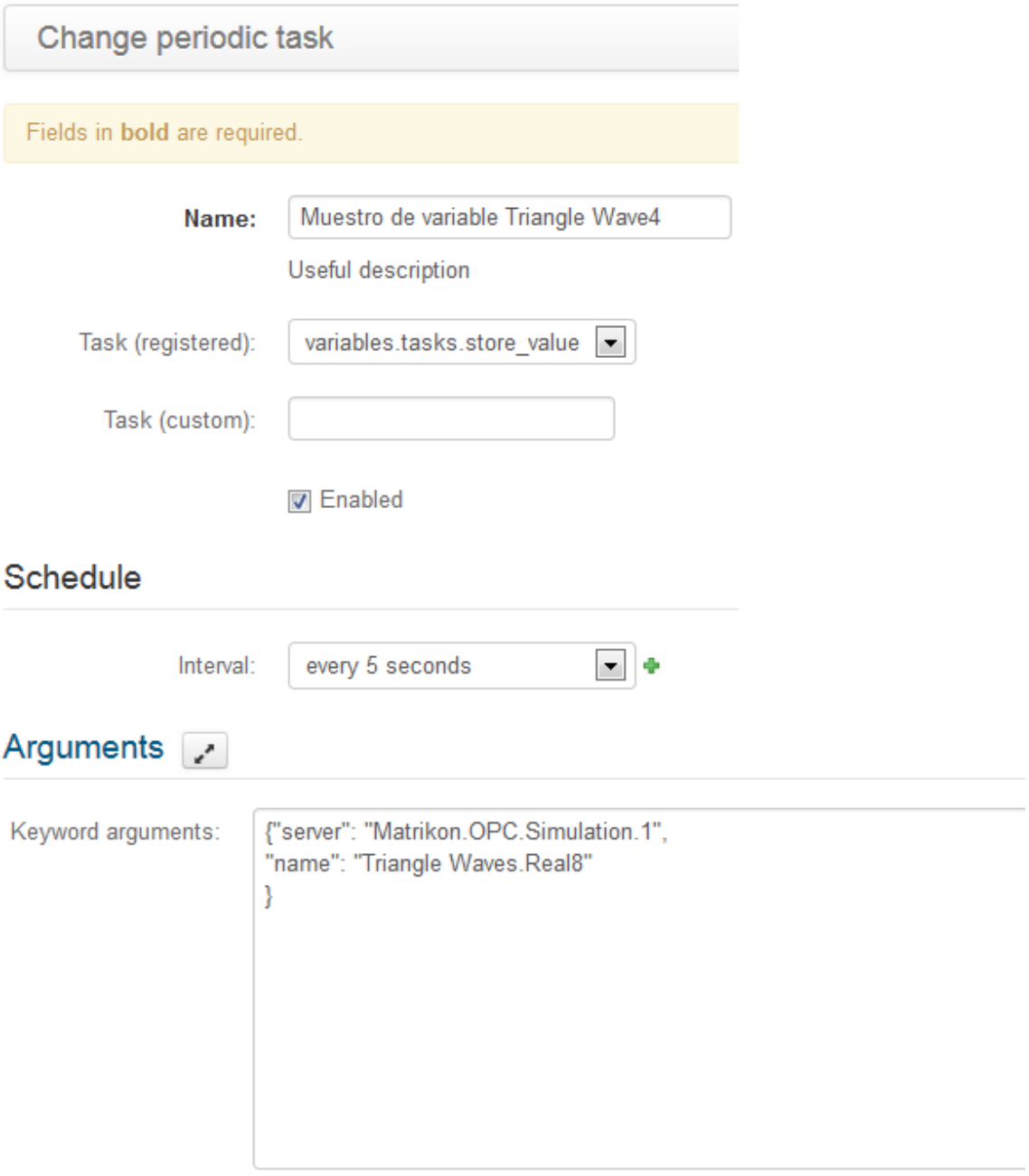


Figura 27 Formulario para la configuración de una actividad periódica en el navegador web.

El formulario que ha rellenado el usuario es enviado a la aplicación web que ejecutará, en este caso, la tarea de almacenar el valor de la variable a la frecuencia estipulada a través del mecanismo de programación Celery. La tarea leerá el valor de la variable y almacenará en la base de datos el valor de la variable y su servidor al igual que la fecha y tiempo de lectura.

```
@task()
def store_value(server,name):
    value,quality,access_rights,time=connect.read(server,name)
    t = TimeSeries(server=server, name=name, value=value, time = datetime.datetime.now()
    t.save()
    return
```

Figura 28 Código de la tarea de guardado del valor de una variable.

En la siguiente figura se puede observar la tabla de la base de datos que se obtiene con la tarea periódica de guardado periódico. La frecuencia de muestro para este ejemplo es de cinco segundos aunque el muestro fue parado hasta en tres ocasiones:

Servidor	Nombre	Valor	Fecha y hora
Matrikon.OPC.Simulation.1	Triangle Waves.Real8	3.14159268452	2013-10-21 17:35:47
Matrikon.OPC.Simulation.1	Triangle Waves.Real8	6.28318536904	2013-10-21 17:35:52
Matrikon.OPC.Simulation.1	Triangle Waves.Real8	9.42477805356	2013-10-21 17:35:57
Matrikon.OPC.Simulation.1	Triangle Waves.Real8	12.5663707381	2013-10-21 17:36:02
Matrikon.OPC.Simulation.1	Triangle Waves.Real8	15.7079634226	2013-10-21 17:36:07
Matrikon.OPC.Simulation.1	Triangle Waves.Real8	25.1327414762	2013-10-21 17:54:44
Matrikon.OPC.Simulation.1	Triangle Waves.Real8	25.1327414762	2013-10-21 17:54:44
Matrikon.OPC.Simulation.1	Triangle Waves.Real8	28.2743341607	2013-10-21 17:54:48

Figura 29 Base de datos del muestro periódico de una variable.

Cuando el usuario desee acceder a esta información para poder representar en su navegador web deberá mediante un formulario solicitar a la aplicación web que variable y que rango de tiempo de tiempo desea consultar:

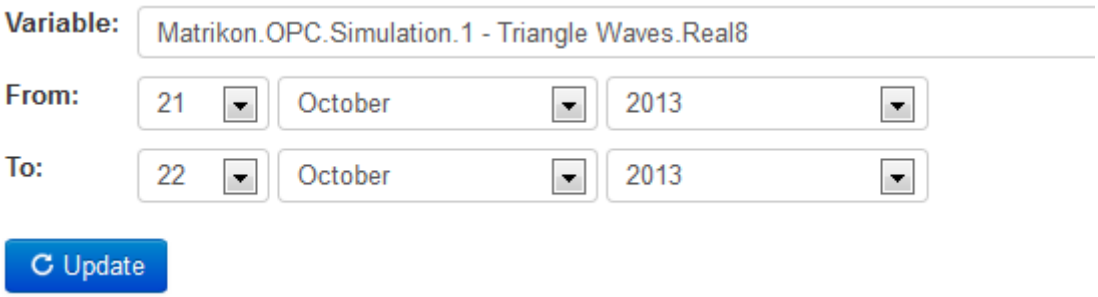


Figura 30 Formulario web para la solicitud de rango de valores de una variable durante un intervalo de tiempo.

Cuando la aplicación web recibe la petición del usuario, utiliza los valores del formulario para filtrar la información que haya en la base de datos y encontrar de esta manera los valores guardados para el rango de tiempo determinado.

Las consultas a la base de datos se realizan mediante una capa de abstracción para bases de datos (del inglés Database Abstraction Layer) que proporciona Django. Esto es una API para facilitar la comunicación mediante la aplicación web python y el lenguaje propio de la base de datos, SQLite en este caso.

En la siguiente figura se puede observar un trozo de la función que realiza lo descrito, más concretamente las líneas que filtran toda las entradas de la base de datos para que solo estén disponibles únicamente las del servidor y variable que ha solicitado el usuario. La función continúa filtrando el resto de parámetros hasta quedarse únicamente con las entradas correctas.

```
@login_required
def json_query_db(request):

    #Check if data sent by user is ok
    data=check_json_data(request)

    #From parameters sent by user, get the objects from database.
    #The format is a database format
    response_json_data={"TimeSeries":[]}
    q=TimeSeries.objects.all()
    q=q.filter(server=data['server'])
    q=q.filter(name=data['name'])
```

Figura 31 Trozo de código de la función que filtra las entradas de la base de datos.

Finalmente la función devuelve en formato JSON al navegador web la información de la variable solicitada, si la hubiese, para que este la procese y la muestre en gráficas o en tablas.

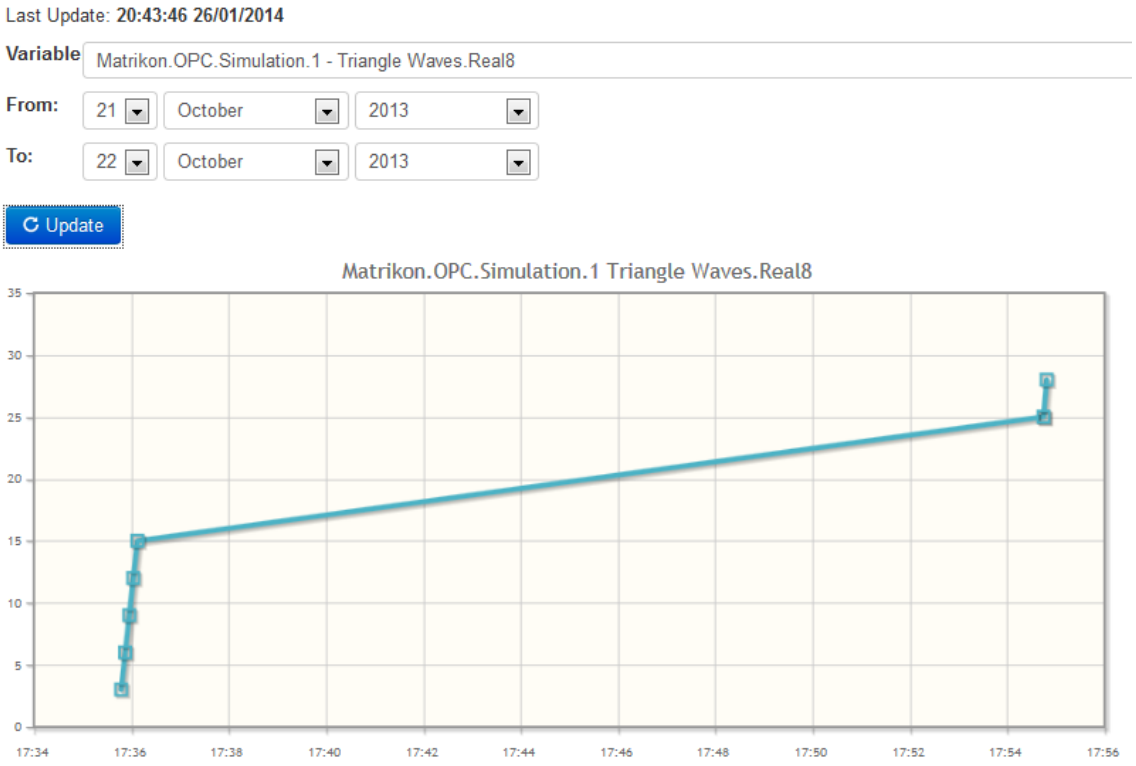


Figura 32 Gráfica de la evolución temporal de una variable a partir de la base de datos.

3.5.5 Actividades periódicas: Alarmas.

El SCADA contiene una función de alarma para que alertar al usuario de que una variable ha alcanzado un valor no deseado. Para que esto sea posible ha de especificarse que variable hay que controlar, el rango de valores que se consideran peligrosos y la frecuencia de muestro de la variable.

Como se puede observar se comparte una similitud con respecto al guardado periódico de variables y es que aquí también existe una componente de periodicidad. Aunque en este caso no se guardan valores en una base de datos.

Por lo tanto el usuario deberá rellenar el mismo formulario que para el guardado periódico solo que en esta ocasión se especificará que la función a repetir ha de ser la encargada de gestionar las alarmas y los argumentos serán: la variable a controlar, el servidor OPC donde se encuentra, el valor límite, la condición para que salte la alarma y el email donde se enviará la notificación.

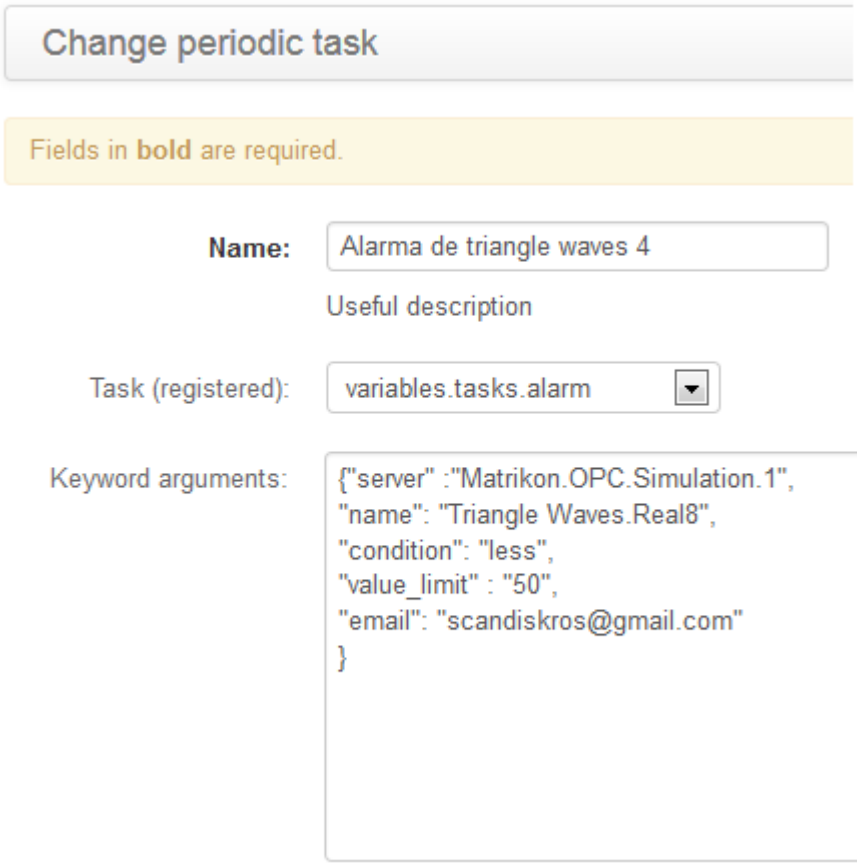


Figura 33 Formulario de tarea periódica empleado para alarmas.

Cuando a la aplicación web le llega la petición del formulario ejecuta, a través de Celery, la tarea periódica encargada de las alarmas. Esta tarea lee el valor de la variable a controlar y si el valor actual no está dentro del rango seguro se envía un email alertando de lo ocurrido.

En la Figura 34 se recoge un trozo del código de la tarea “alarm”. Esta función recibe los valores del formulario como parámetros. A continuación lee el valor de la variable y evalúa si está dentro del rango que ha especificado el usuario para mandar un email.

```
@task()
def alarm(server,name,condition,value_limit,email):
    #reading values from opc client
    value,quality,access_rights,time=connect.read(server,name)
    #print value, value_limit,float(value)

    if condition=='greater':
        value_limit=float(value_limit)#converting the variable from string to float
        value=float(value)     #converting the variable from string to float
        if  value > value_limit:
            send_mail('Online Scada Alarm', 'The variable ' + str(name) + ' on serve
                      '. The actual value is '+str(value)+'.','onlinescadapfc@gmail.
```

Figura 34 Código de la tarea encargada de gestionar las alarmas del SCADA.

A continuación un ejemplo del email de notificación que recibiría el usuario:

Figura 35 Email de notificación de alarma ocurrida en el SCADA.

4. Capítulo 4. Resultados

El resultado de este proyecto ha sido la creación de una aplicación web que cumple con los objetivos presentados en el Capítulo 1. Es accesible remotamente gracias al servidor Apache y la aplicación web python, existe un acceso basado en credenciales de usuario gracias a las funciones integradas de Django y se consigue un control universal de PLCs gracias al protocolo OPC.

Para conseguir montar la plataforma ha de instalarse en un PC, que actúa a modo de servidor, los siguientes programas o paquetes:

- Apache: Como servidor de internet con comunicación HTTP.
- Python: Como lenguaje de programación de la aplicación web.
- Django: Como framework de la aplicación web.
- Celery: Para la programación automática de tareas.
- Django_admin_bootstrapped: Configura ciertos ficheros de Django para cambiar la estética de alguna página web.
- Servidor OPC: Que el cliente final del proyecto elegirá según sus necesidades.

Adicionalmente, se deberán almacenar los siguientes ficheros estáticos en el servidor para facilitárselos al usuario cuando éste los requiera:

- Documentos HTML con las plantillas de cada página web.
- Ficheros con librerías JavaScript y CSS para la funcionalidad de las páginas web.

La estructura del sitio web desarrollado se explicará a continuación para dar una noción de cómo ha quedado el SCADA. Primer se mostrará una figura a modo de esquema para dar un conocimiento global para después entrar en detalle con cada una de las páginas web.

El sitio web construido es un conjunto de páginas web que forman lo que hemos llamado, SCADA Online, concepto que da título al presente proyecto.

Las páginas web se pueden dividir en cuatro aparatados: uno para acceso a datos de las variables de los PLCs, su supervisión y su control, el siguiente como acceso histórico a la base de datos que contiene información guardada de las variables a controlar, un tercero para alarmas y eventos que puedan surgir acerca de las variables de los PLCs y finalmente un conjunto de sitios para la administración de la aplicación web desde el navegador web.

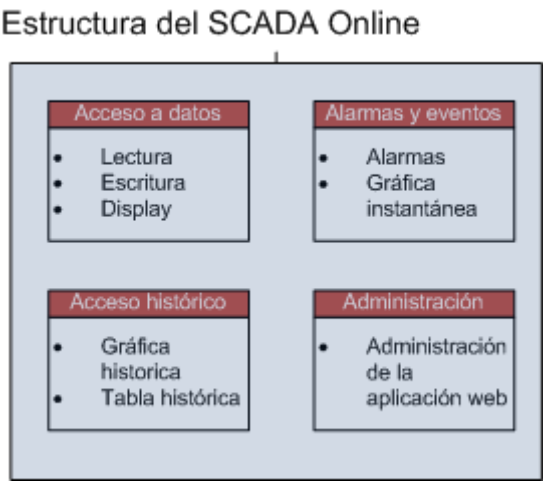


Figura 36 Estructura del sitio web creado para el SCADA Online.

A continuación se comentarán las páginas web creadas para cada uno de los apartados explicados detallados arriba.

Páginas web para el acceso a datos:

- Lectura: con este página se crea una tabla con las variables que controla el usuario para poder sus valores. Se dispone de unos botones para actualizaciones periódicas de la tabla.

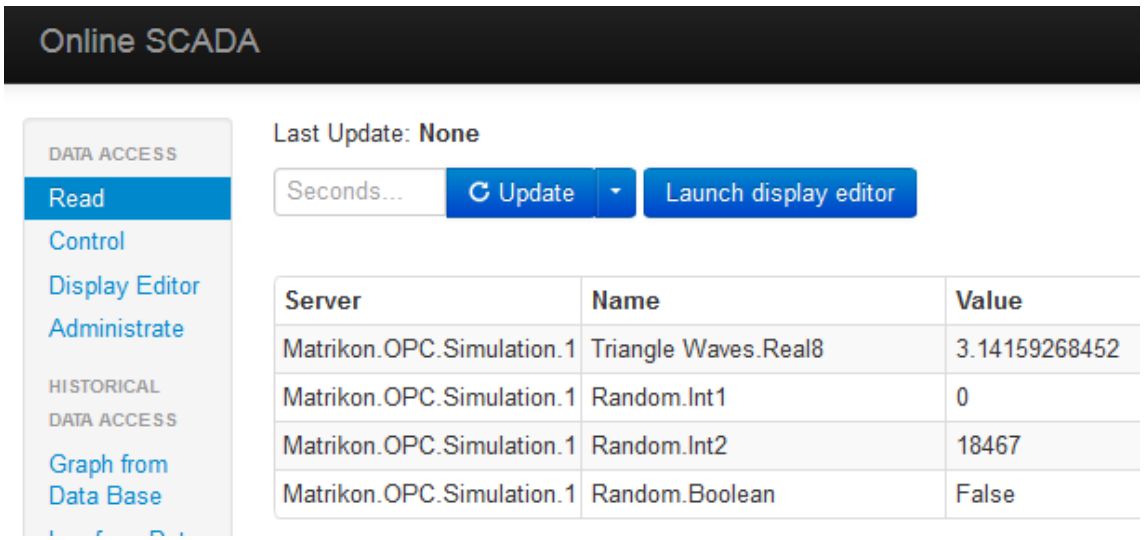


Figura 37 Página web para lectura de variables.

- Control: a través de una tabla con formulario se puede escribir sobre las variables de los PLCs que tenga habilitada esa posibilidad. Se le informa al usuario mediante alertas de color verde o rojo de que la operación de escritura ha sido exitosa o no:

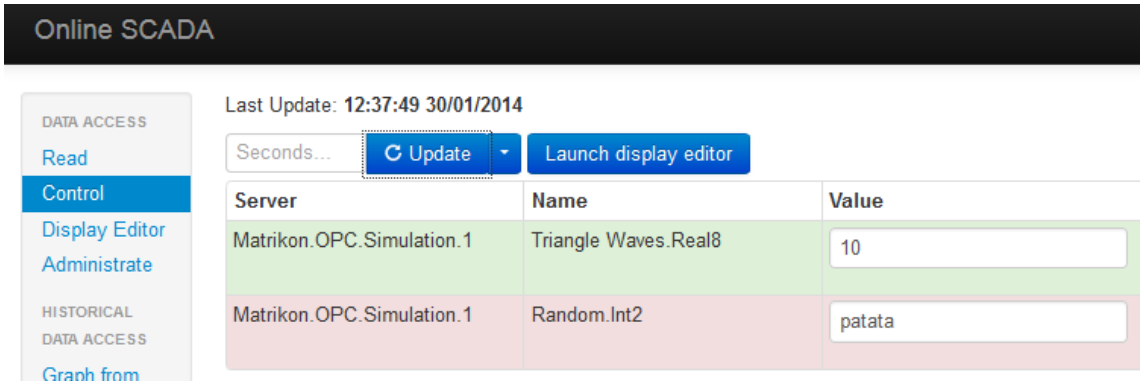


Figura 38 Página web para escritura de variables.

- Editor de display: se selecciona cuáles de las variables que controla el usuario estarán visibles en las páginas web del SCADA. Es probable que el usuario no desee controlar todas las variables al mismo momento.

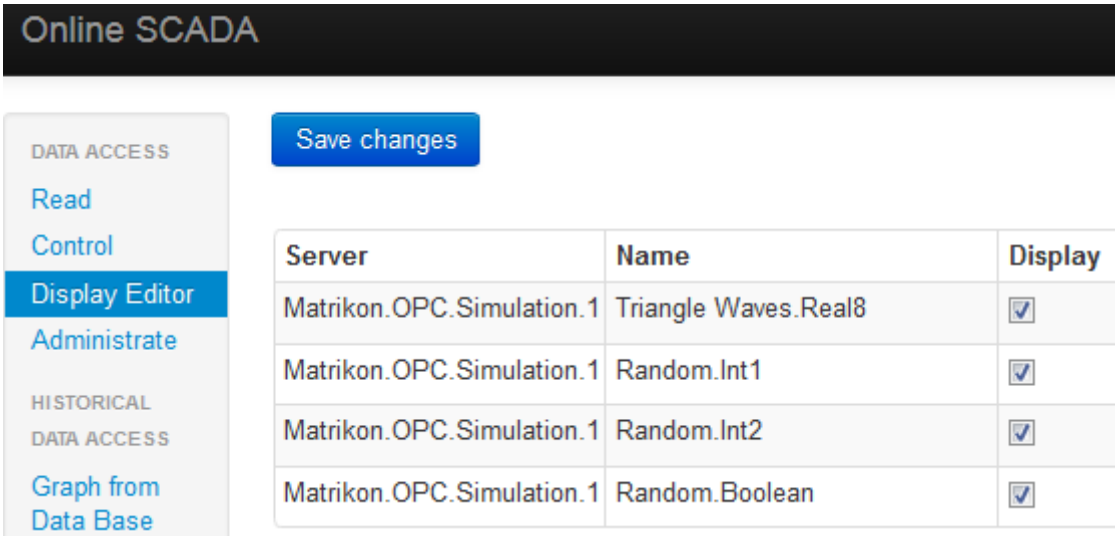


Figura 39 Página web para elección de variables a controlar

Páginas web para acceso histórico

- Gráfica desde la base de datos: s eleccionando una variable y un rango de tiempo determinado se mostrará una gráfica de la evolución de esa variable si estuviese registrada en la base de datos.

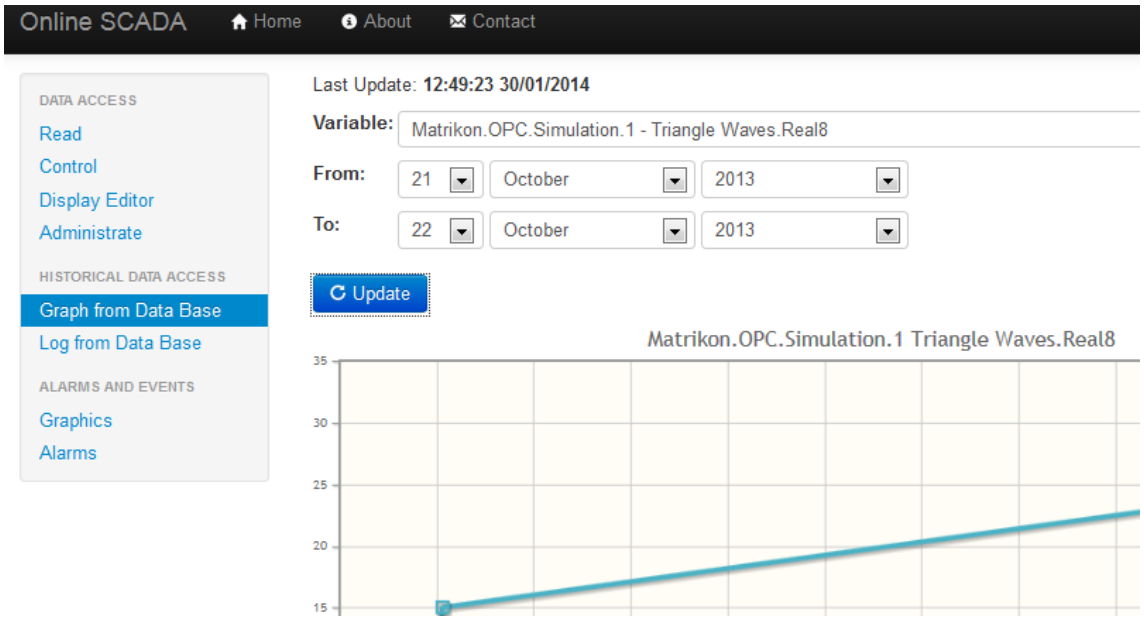


Figura 40 Página web para la representación gráfica del registro histórico de una variable.

- Registro histórico: de la misma forma que la página web anterior pero esta vez no se representa mediante una gráfica sino que se le da al usuario una tabla con los valores registrado de esa variable para que se pueda exportar a otra plataforma que requiera el usuario.

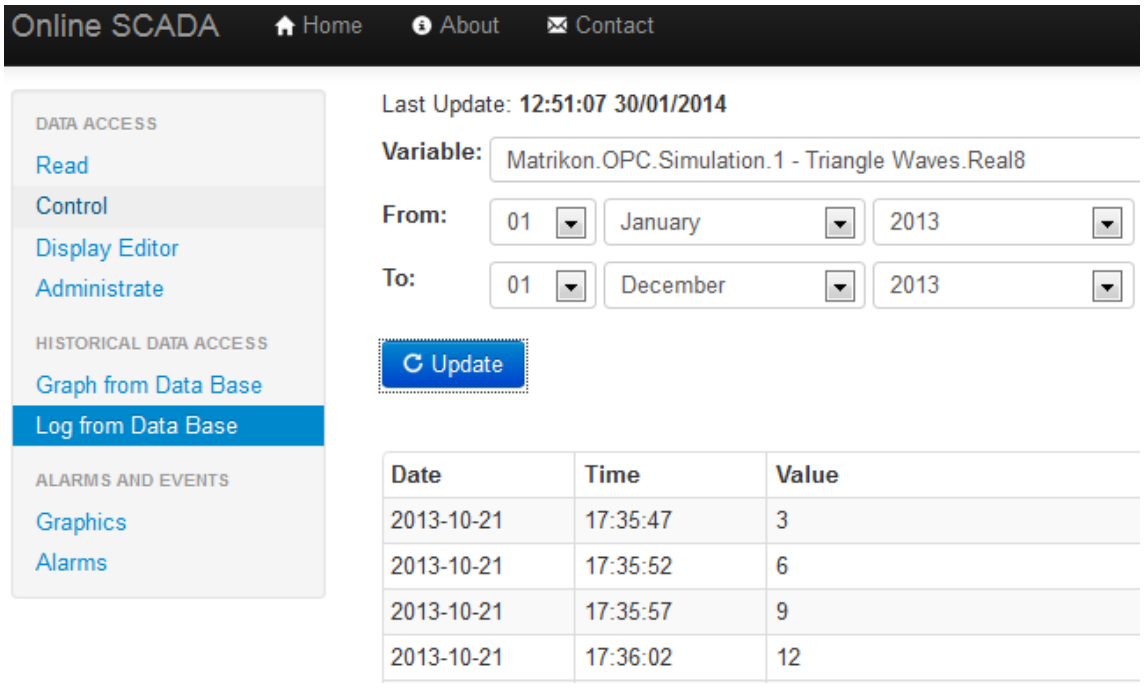


Figura 41 Página web para el muestro en tabla de la evolución histórica de una variable.

Alarmas y eventos

- Gráficas: para representar una variable se puede crear una gráfica que nos haga una visualización en directo del estado de la variable con una frecuencia de actualización configurable.

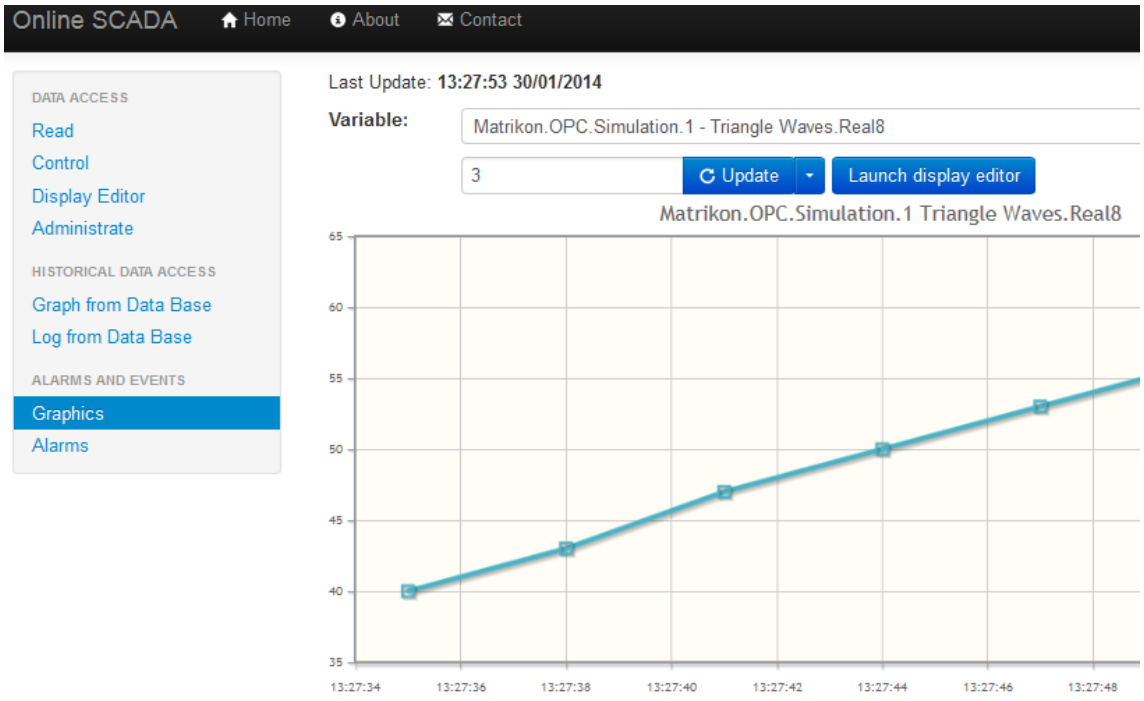


Figura 42 Página web para la creación de una gráfica a tiempo real del estado de una variable

- Alarmas: mediante un panel administrativo el usuario puede configurar notificaciones a su email para alertar sobre el valor no deseado que pueda tomar una variable

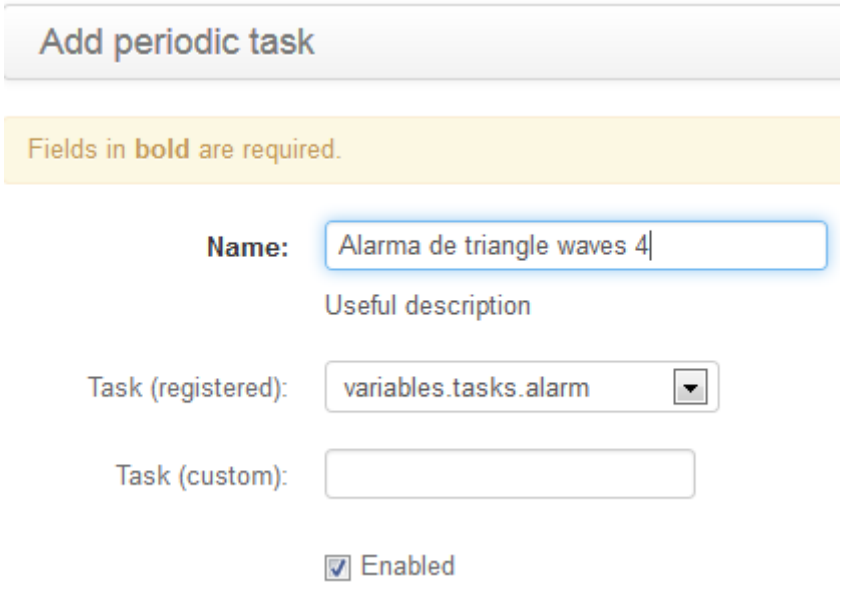


Figura 43 Página web para creación de alarmas configurables por el usuario.

Acceso para administrador:

- Página para administrador: si bien esta página no forma parte del SCADA, en cuanto a que no contribuye a controlar los PLCs, es necesaria para el funcionamiento de la página web. La página web sigue siendo una plataforma software que necesita ser

configurada por algún usuario con una serie de conocimientos. Desde esta página se podrán controlar tareas administrativas como por ejemplo que usuarios pueden acceder al SCADA o que pueden realizar esos usuarios en el SCADA.

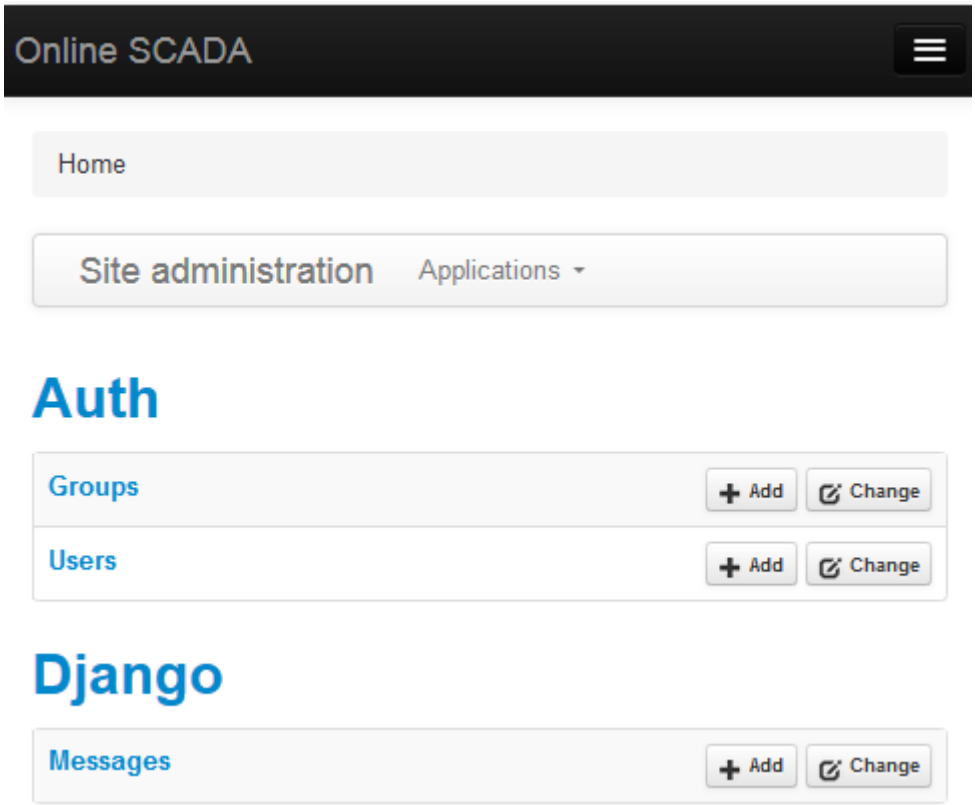


Figura 44 Página web para la administración de la aplicación web desde el navegador.

5. Capítulo 5. Conclusiones

En primer lugar me gustaría agradecer la oportunidad de haber realizado este proyecto fin de carrera a la Cátedra TAIM WESER – Universidad de Zaragoza

Para llevar a cabo el proyecto, partí sin mucha experiencia previa en la programación o comunicación con PLCs, siendo ambas las piedras angulares del SCADA que se ha desarrollado. Mientras realizaba el estudio previo para poder asegurar la comunicación universal de marcas de PLC tuve siempre la incertidumbre de si podría conseguirlo. Algo a lo que no me había enfrentado antes, pues en la carrera si uno se prepara una asignatura, sabe que conseguirá aprobarla.

Otra tarea importante del proyecto fue la de trabajar en diversos entornos de programación para conseguir el SCADA online. Desde entornos gráficos, comunicación con internet, programación de servidores web y desarrollo de aplicaciones web. Todo esto ha supuesto una curva de aprendizaje lenta, pues lo trabajado no es el enfoque natural de mi carrera, pero que ha concluido en este proyecto multidisciplinar en el que se han cubierto todos los objetivos propuestos: la supervisión y control de cualquier marca de PLCs de manera remota.

Por otra parte y en referencia al SCADA, pienso que el hecho de que lo haya decidido desarrollar para funcionar en un navegador web frente a realizar una aplicación de escritorio ha sido algo importante. Últimamente, en el mundo de la informática, se está tendiendo a la computación en internet. Hace unos años esto se traducía en almacenamiento de datos online, pero ahora es fácil ver plataformas de desarrollo de software que funcionan completamente Online como, "Salesforce.com Inc", o de computación online de tareas.

Por esto, este SCADA marca una diferencia frente a otro SCADA hecho mediante una aplicación de escritorio. Pues no en tanto en las funciones que pueda realizar como SCADA propiamente dicho, sino en la manera de ejecutarse, que es mediante internet.

6. Anexos

A lo largo de la Memoria se ha ido explicando el funcionamiento del SCADA Online sin entrar con mucho detalle en el código que lo desarrolla. Por lo tanto, aquí se va a exponer y explicar con detalle el código desarrollado únicamente por el autor para este SCADA y así quedará clara para el lector la frontera entre lo que ha creado el autor y las librerías integradas en los distintos frameworks usados.

6.1 Código del servidor web

En este apartado se mostrará y explicará el código relacionado con las funciones que componen la aplicación web.

6.1.1 Fichero urls.py

Este fichero, *urls.py,* alojado dentro de la carpeta *django_scada* sirve para relacionar cada una de las urls del proyecto con su función python.

Esta es la estructura de urls creada por el autor donde mediante la directiva *url(r'^home/', 'variables.views.home')* se ha asociado a la url *home/* la función *home* que está en el fichero *views* dentro de la carpeta *variables* . Esta directiva es empleada con el resteo de las urls del sistema como se ve en el siguiente conjunto de código.

```
urlpatterns = patterns('',
    #Url de la página de inicio.
    url(r'^home/', 'variables.views.home'),
    #Url del manual de usuario
    url(r'^manual/', 'variables.views.manual'),
    #Url de lecutras de vairable
    url(r'^read/', 'variables.views.read'),
    #Url para enviar datos mediante HTTP Request para refrescar variables
    url(r'^json/read_update/','variables.views.read_update'),
    #Url de escritura de variables
    url(r'^control/','variables.views.control'),
    #Url para enviar datos mediante HTTP Request para escribir variables
    url(r'^json/control_response/',  'variables.views.control_response'),
    #Url para selecionar las variables a mostrar
    url(r'^display_editor/','variables.views.display_editor'),
    #Url para intercambio de datos cliente-navegador mediante HTTP Request
    url(r'^json/display_editor_response/',
'variables.views.display_editor_response'),
    #Url para la página con la gráfica para variables
    url(r'^graph_live/', 'variables.views.graph_live'),
    #Url de intercambio asíncrono de datos para la gráfica mediante HTTP
Request
    url(r'^json/graph_live/',  'variables.views.json_graph_live'),
    #Url para mostrar la gráfica de representación histórica de variables
    url(r'^graph_db/', 'variables.views.graph_db'),
    #Url para intercambio de datos mediante HTTP Request de la gráfica
histórica
    url(r'^json/graph_db/',  'variables.views.json_query_db'),
    #Url para crear una tabla con los valores históricos de variables
    url(r'^log_db/', 'variables.views.log_db'),
    #Url para intecambio de datos mediante HTTP Request para la tabla de
valores históricos
    url(r'^json/log_db/',  'variables.views.json_query_db'),
    #Url de django para el menú de administrador
```

6.1.2 Fichero automataconnect.py

Con las funciones de este fichero se comunican las funciones de la aplicación web con el cliente OPC y de esta manera se puede considerar este fichero como una librería

La función *read* tiene como parámetros de entrada el servidor y el nombre de la variable a leer. Con esta información se conecta al servidor indicado y hace una letura para el valor, calidad , fecha a la que se ha leído la variable y además los derechos de acceso a esa variable. Finalmente devuelve a la función de la que ha sido llamado todos los datos en formato *string.* En caso de que no se haya podido realizar la lectura, devuelve *strings* avisando del error.

```
def read(server,name):
    try:
        opc = OpenOPC.open_client('localhost')
        opc.connect(server)
        value, quality, time = opc.read(name)
        access_rights=opc.properties(name, id=5)
        opc.close()
        value=str(value)
        return value,quality,access_rights,time

    except Exception:
        opc.close()
        return None,'Error',None,None
```

La función *write*, funciona de manera similar a la función *read.* Tiene como parámetros de entrada el servidor, nombre de la variable, valor de escritura y un vector. Una vez conectado al servidor indicado se realiza la escritura de variable. Finalmente añade al vector si la operación de escritura ha sido exitosa o no, y se devuelve ese vector a la función que hubiera llamado a la función *write.*

```
def write(server,name,value,response_array):
    try:
        opc = OpenOPC.open_client('localhost')
        opc.connect(server)
        #Si la operación de lectura ha sido un exito
        if value:
            a=opc.write((name,value))
        else:
            a=''#No se hace nada si el usuario no escribe nada
        response_array.append(a.lower())
        return response_array

    except Exception:
        response_array.append('error')
        return (response_array)
```

6.1.3 Fichero tasks.py

En este fichero se encuentran las tareas programables del sistema. Estas son las tareas que se pueden automatizar y ejecutar de manera periódica.

La tarea *store_value* permite almacenar lecturas de las variables de los PLCs en la base de datos del sistema. Como parámetros de entrada tiene la variable a guardar y el servidor donde está alojada. Las operaciones son una lectura de la variable, calidad, derechos de acceso y fecha de lectura mediante una llamada a la función *read* del fichero *automataconnect.py*. Los datos de lectura son alojados en una variable que funciona modelo *TimeSeries* y finalmente se guarda el valor en la base de datos.

```
      @task()
def store_value(server,name):
    value,quality,access_rights,time=connect.read(server,name)
    t = TimeSeries(server=server, name=name, value=value, time =
datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
    t.save()
    return
```

La tarea *alarm* tiene como función avisar al usuario mediante un email de que una variable ha salido del rango que él ha fijado. Como parámetros de entrada están el servidor, nombre de la variable, el tipo de condición para activar la alarma, el valor límite y el email al que mandar el aviso. Se realiza una lectura de la variable indicada y se evalúa el valor actual con el valor límite según la condición de activación de alarma dada. Si la alarma fuera activada se enviaré un email con los datos de la variable, el servidor y se indicaría que la evaluación del valor actual ha desencadenado la alarma. Las condiciones de activación de alarma son: mayor que, menor que, igual y no igual.

```
@task()
def alarm(server,name,condition,value_limit,email):

    value,quality,access_rights,time=connect.read(server,name)    #Lectura
del valor del ciente opc

    if condition=='greater':
        value_limit=float(value_limit)     #La variable se convierte de
string a float
        value=float(value)     #La variable se convierte de string a float
        if  value > value_limit:
            send_mail('Online Scada Alarm', 'The variable ' + str(name) + '
on server ' + str(server) + ' is greater than its value limit '+
str(value_limit) +
                      '. The actual value is
'+str(value)+'.','onlinescadapfc@gmail.com',[str(email)], fail_silently=False

    if condition=='less':
        value_limit=float(value_limit)
        value=float(value)

        if  value < value_limit:
            send_mail('Online Scada Alarm', 'The variable ' + str(name) + '
on server ' + str(server) + ' is lesser than its value limit '+
str(value_limit)+
                      '. The actual value is
'+str(value)+'.','onlinescadapfc@gmail.com',['scandiskros@gmail.com'],
fail_silently=False)

    if condition=='equal':
        value_limit=float(value_limit)
        value=float(value)
        if  value == value_limit:
            send_mail('Online Scada Alarm', 'The variable ' + str(name) + '
on server ' + str(server) + ' is equal to its limit '+ str(value_limit),
                      'onlinescadapfc@gmail.com',['scandiskros@gmail.com'],
fail_silently=False)

    if condition=='not equal':
        value_limit=float(value_limit)
        value=float(value)
        if  value != value_limit:
            send_mail('Online Scada Alarm', 'The variable ' + str(name) + '
on server ' + str(server) + ' is not equal to '+ str(value_limit)+
                      '. The actual value is
'+str(value)+'.','onlinescadapfc@gmail.com',['scandiskros@gmail.com'],
fail_silently=False)
    return
```

6.1.4 Fichero *views.py*

Este fichero contiene las funciones que han de ejecutarse cuando el usuario accede a la url asociada.

Existe un conjunto de funciones cuya única misión es facilitar el documento HTML al cliente. Como parámetro de entrada tienen el contenido de la petición HTTP que ha mandado el cliente al servidor, las funciones *home* y *manual* devuelven los ficheros *home.html* y *manual.html* respectivamente cuando son llamadas.

```
@login_required
def home(request):
    return
render_to_response('home.html',context_instance=RequestContext(request))

@login_required
def manual(request):
    return
render_to_response('manual.html',context_instance=RequestContext(request))
```

Otro conjunto de funciones realiza una operación similar pero además añaden un diccionario con información relativa a las variables de los PLCs que se están controlando. De los datos que contiene ese diccionario se rellenarán las distintas tablas o gráficas de las páginas webs.

```
@login_required
def read(request):
    variable_list=get_variables(request.user)
    return render_to_response('read.html', {'variable_list': variable_list} ,
context_instance=RequestContext(request))

@login_required
def control(request):
    variable_list=get_writable_variables(request.user)
    return render_to_response('control.html', {'variable_list': variable_list} ,
context_instance=RequestContext(request))

@login_required
def display_editor(request):
    variable_list=get_variables(request.user)
    return render_to_response('display_editor.html', {'variable_list':
variable_list} , context_instance=RequestContext(request))

@login_required
def graph_live(request):
    variable_list=get_variables(request.user)
    return render_to_response('graph_live.html', {'variable_list':
variable_list} , context_instance=RequestContext(request))


@login_required
def graph_db(request):
    variable_list=get_variables(request.user)
    return render_to_response('graph_db.html', {'variable_list': variable_list}
context_instance=RequestContext(request))

@login_required
def log_db(request):
    variable_list=get_variables(request.user)
    return render_to_response('log_db.html', {'variable_list': variable_list} ,
context_instance=RequestContext(request))
```

Para conseguir el diccionario con los valores actuales de las variables se utiliza en todas las funciones enunciadas hasta ahora la función *get_variables.* Esta función tiene como parámetro de entrada el usuario que ha mandado la petición HTTP a la aplicación web, y crea una lista de todas las variables que ha guardado ese usuario llamando a la función *variable_list_django_db*, que está explicada más adelante. Una vez creada la lista de variable, se crea otra con los valores de las operaciones de lectura, que es la que finalmente se devuelve como parámetro de salida.

```
def get_variables(user):
    variable_list=variable_list_django_db(user)
    i=0
    while i<len(variable_list):

variable_list[i].value,variable_list[i].quality,variable_list[i].access_rights,t
ime=connect.read(variable_list[i].server,variable_list[i].name)
        i=i+1
    return variable_list
```

En la función *variable_list_django_db*, que tiene como parámetro de entrada el usuario que ha realizado la petición HTTP se realiza una consulta a la base de datos para extraer todas las variables almacenadas por ese usuario. Las variables son filtradas de tal manera que si el usuario hubiera introducido la misma variable más de una vez por error el sistema lo detecta y no muestra la variable por duplicado en el SCADA.

```
def variable_list_django_db(user):

    server_list=[]
    raw_variable_list=Variable.objects.filter(user=user)
    variable_list = []
    for e in raw_variable_list:
        i=0
        encontrado=False
        while i<len(variable_list):
            if variable_list[i].server==e.server and
variable_list[i].name==e.name:
                encontrado=True
                break
            else:
                encontrado=False
                i=i+1
        if not encontrado:
            variable_list.append(e)

    return variable_list
```

La función *control_response* se utiliza para generar los datos de vuelta cuando el usuario decide enviar una petición HTTP parar escribir sobre las variables de los PLCs. Como parámetro de entrada tiene el cuerpo de la petición HTTP, en este cuerpo se encuentran los datos de las variables a modificar y los valores de escritura. La función verifica si el usuario ha mandado los datos de forma correcta y escribe llamando a la función *write* del fichero *automataconnect.py* en los PLCs. Finalmente, se envía un vector codificado en formato json donde se indica si las operaciones de escritura se han realizado con éxito.

```
@permission_required('variables.change_Variable',raise_exception=True)
def control_response (request):
    #Verifica si los datos de la petición HTTP son correctos.
    if request.method == 'POST':
        json_data = simplejson.loads(request.body)
    try:
        data = json_data['data'][0]
    except KeyError:
        return HttpResponseServerError("Malformed data!")
    #Crea vector con datos de respuesta
    response_array=[]
    i=0
    while i<len(json_data['data']):

server,name,value=json_data['data'][i]['server'],json_data['data'][i]['name']
,json_data['data'][i]['value']
        response_array=connect.write(server,name,value,response_array)
        i=i+1
    return HttpResponse(json.dumps(response_array),
mimetype='application/json')
```

La función *read_update* devuelve, ante una petición HTTP del usuario, los datos actualizados de lectura de los PLCs. Como parámetro de entrada está el cuerpo de la petición HTTP conteniendo los datos de las variables a leer. Cuando la función ha realizado las operaciones de lectura de todas las variables se crea un diccionario que se codifica en formato json para enviarlo al usuario.

```
@login_required
def read_update(request):
    json_data={"variables":[]}
    variable_list=get_variables(request.user)
    for e in variable_list:
        dict={}
        dict['server']=e.server
        dict['name']=e.name

dict['value'],dict['quality'],dict['access_rights'],time=connect.read(e.serve
r,e.name)
        json_data['variables'].append(dict)
    return HttpResponse(json.dumps(json_data), mimetype='application/json')
```

La función *json_query_db* es una función que realiza una consulta a la base datos para extraer los valores históricos de una variable determinada durante un intervalo de tiempo concretado por el usuario para finalmente devolver un diccionario python con los valores requeridos. El parámetro de entrada es una petición HTTP, en cuyo cuerpo se encuentran los datos de la consulta: variable y su servidor, fecha de inicio de búsqueda y fecha de fin de búsqueda. Cuando la función ha realizado una consulta, donde se extraen todos los valores históricos de todas las variables almacenadas, se filtran los resultados en función de los parámetros que ha indicado el usuario. Y finalmente se transforma la variable, que se encuentra en un formato propio de la base de datos a un diccionario python.

```
@login_required
def json_query_db(request):

    data=check_json_data(request)
    #Consultas a la base de datos y filtrado de los datos obtenidos.
    response_json_data={"TimeSeries":[]}
    q=TimeSeries.objects.all()
    q=q.filter(server=data['server'])
    q=q.filter(name=data['name'])

    to_year=int(data['to-year'],base=10)
    to_month=int(data['to-month'],base=10)
    to_day=int(data['to-day'],base=10)

    from_year=int(data['from-year'],base=10)
    from_month=int(data['from-month'],base=10)
    from_day=int(data['from-day'],base=10)

    q=q.exclude(time__gte=datetime.datetime(to_year,to_month,to_day))
    q=q.filter(time__gte=datetime.datetime(from_year,from_month,from_day))
    q=q.order_by('time')
    #Transfomar la variable que se encuentra en un formato propio de la base
de datos a un diccionario python.
    if not q:
        dict={}
        dict['value']='None';
        dict['time']="None";
        response_json_data['TimeSeries'].append(dict)
    else:
        for e in q:
            print e.server,e.name,e.value,e.time

            dict={}
            dict['value']=e.value
            dict['time']=e.time
            response_json_data['TimeSeries'].append(dict)
    return HttpResponse(json.dumps(response_json_data, cls=DjangoJSONEncoder)
mimetype='application/json')
```

6.2 Código del cliente

En este apartado se mostrará y explicará el código que se ejecuta en la parte del cliente, y que compone el conjunto de funciones javascript que usa el interfaz gráfico. Estas funciones son: *get_variable, process_value, graph_plot.* Todas están en lenguaje JavaScript y se encuentran en los documentos HTML que se alojan en el dispositivo del usuario final. Las funciones son llamadas mediante botones cuando se quieren actualizar variables, escribir variables, y dibujar una gráfica en tiempo real.

6.2.1 Función get_variable

La función *get_variable* es usada en la página web de lectura de variables y sirve para refrescar los valores de las variables mediante comunicación asíncrona con el servidor. Carece de parámetros de entrada y utiliza un spinner para indicar al usuario que la función se está ejecutando. Mediante el uso de la función jQuery *$.getJSON*, realiza una comunicación asíncrona con el servidor web y este devuelve una variable en formato json con los nuevos valores de las variables. A continuación y mediante selectores jQuery, se eliminan los antiguos

valores de la tabla HTML de la página web para introducir los datos nuevos. Finalmente se muestra la hora de la última actualización para informar al usuario.

```
function get_variable(){
  //Activa el spinner para indicar al usuario que se esta ejecutando la función
  $('#date').empty();
  $('#date').append(spinner.el);
  //Función para comunicación asíncrona a la url "/json/read_update" para refrescar lectura
  $.getJSON("/json/read_update/", function(json_data) {
    //El dato "json_data" contiene los nuevos valores de las variables devueltos por el servidor
    var n = $("tr").length;
    for (var i=0; i<n-1; i++){
      //Mediante selectorse jQuery se modifica el aspecto de las tablas si ha habido algún error.
      if (json_data.variables[i].quality=='Error'){
        $("#variable"+i).removeClass();
        $("#variable"+i).toggleClass('error');
      }
      //Mediante selectorse jQuery se elimina el antiguo valor de la tabla HTML y se añade el
nuevo.
      $("#variable"+ i + " td#server").empty();
      $("#variable"+ i + " td#server").append(json_data.variables[i].server);

      $("#variable"+ i + " td#name").empty();
      $("#variable"+ i + " td#name").append(json_data.variables[i].name);
      if (json_data.variables[i].quality=='Error'){
        $("#variable"+ i + " td#value").empty();
        $("#variable"+ i + " td#value").append('<i class="icon-exclamation-sign"></i>');
      }
      else{
        $("#variable"+ i + " td#value").empty();
        $("#variable"+ i + " td#value").append(json_data.variables[i].value);
```

```
    }


  }
  //Pongo fecha de la ultima actualización
  var myDate = new Date();
  var seconds=myDate.getSeconds();
  var minutes=myDate.getMinutes();
  var hours=myDate.getHours();

  if (seconds<10) {seconds=('0'+seconds);}
  if (minutes<10) {minutes=('0'+minutes);}
  if (hours<10) {hours=('0'+hours);}

  var displayDate =  hours + ':' + minutes +  ':' + seconds + ' ' + (myDate.getDate()) + '/' +  ( '0' +
(myDate.getMonth()+1) ).slice( -2 ) + '/' + myDate.getFullYear();
  $('#date').empty();
  $('#date').append('<p>Last Update: <strong>' +  displayDate + '</strong></p>');
 });
};
```

6.2.2 Función process_value

A través de la función *process_value* se mandan al servidor web los datos de escritura de las variables y se procesan los datos que se reciben para ser mostrados en la página web. Primeramente, se utilizan selectores jQuery para manipular la estructura de los elementos del documento HTML y se recoge la información de escritura en un objeto JavaScript que es codificado en formato json. A continuación se envía esta información al servidor web mediante la función $.post de jQuery, que a diferencia de la función $.get usada anteriormente, permite mandar datos al servidor. El servidor manda en una variable llamada “dta” la información si las operaciones de escritura han sido realizadas con éxito. Y finalmente, y mediante selectores jQuery se modifica el documento HTML para informar al usuario del éxito o fracaso de la escritura al igual que la fecha en la que se realizó.

```
function process_value(){
  $('#date').empty();
  $('#date').append(spinner.el);
  var n = $("tr").length;
  //Mediante selecotrse jQuery se crea un objeto JavaScript con la información de las variables a escribir
  response = {"data" : []}
  for (var i=0; i<n-1; i++){
    var obj={};
     $("#variable"+ i + " td#server").each(function(index, domele) {
       obj['server'] = $(this).text();
     });
     $("#variable"+ i + " td#name").each(function(index, domele) {
       obj['name'] = $(this).text();
     });
     $("#variable"+ i + " input:text").each(function(index, domele) {
       obj['value'] = domele.value;
     });
    response.data.push(obj);
  }
  //Codificación de la variable JavaScript en formato json.
  json_response = JSON.stringify(response);
  //Función $.post de jQuery para envío y respuesta de datos al servidor
  $.post("/json/control_response/", json_response).success(function(dta) {
    for (var i=0; i<n-1; i++){
      //"dta" retorna success o error, para cada operación. Informamos al usuario con un cambio en el color de la tabla
     $("#variable"+i).removeClass();
```

```
    $("#variable"+i).toggleClass(dta[i]);
   }
    //Pongo fecha de la ultima actualización
    var myDate = new Date();
    var seconds=myDate.getSeconds();
    var minutes=myDate.getMinutes();
    var hours=myDate.getHours();

    if (seconds<10) {seconds=('0'+seconds);}
    if (minutes<10) {minutes=('0'+minutes);}
    if (hours<10) {hours=('0'+hours);}

    var displayDate =  hours + ':' + minutes +  ':' + seconds + ' ' + (myDate.getDate()) + '/' +  ( '0'
+ (myDate.getMonth()+1) ).slice( -2 ) + '/' + myDate.getFullYear();
    $('#date').empty();
    $('#date').append('<p>Last Update: <strong>' +  displayDate + '</strong></p>');
  });
};
```

6.2.3 Función *graph_plot*

Esta función permite al usuario dibujar una gráfica en tiempo real del estado de una variable determinada. De esta manera su funcionamiento es similar al de la función *process_value,* pues ambas recogen mediante selectores jQuery los datos que el usuario ha seleccionado para, después de codificarse en formato json, enviarlos mediante una función $.post al servidor de internet. Como diferencia está el hecho de que los datos de respuestas del servidor, son alojados en un vector y mostrados por pantalla como una gráfica gracias a las funciones de las librerías de jqPlot.

```
function graph_plot(){
 $('#date').empty();
 $('#date').append(spinner.el);
 //Se recogen los datos que ha seleccionado el usuario
 response = {"data" : []}
  $("select").each(function(index, domele) {
    var obj={};
    obj['server'] = $(this).find('option:selected').attr('id');
    obj['name'] = $(this).find('option:selected').attr('class');
    response.data.push(obj);
  });
 //Se codifican en formato json los datos del usuario
 json_response = JSON.stringify(response);
 //Se verifica si la variable a mostrar ha cambiado.
 if (previous_server != "None"){
  if (!(previous_server == obj['server'] && previous_variable == obj['name'])){
    new_data=true;
  }
 }
 previous_variable = obj['name'];
 previous_server = obj['server'];
 //Llamada al servidor con envío de datos mediante $.post
 $.post("/json/graph_live/", json_response).success(function(dta) {
    //Codigo para añadir el nuevo data al vector de la gráfica y dibujarla
    var value = parseInt(dta.value,10);
    if (new_data){
      line1=[[dta.time,value]];
```

```
    var displayDate =  hours + ':' + minutes +  ':' + seconds + ' ' + (myDate.getDate()) + '/' +  ( '0'
+ (myDate.getMonth()+1) ).slice( -2 ) + '/' + myDate.getFullYear();

    $('#date').empty();

    $('#date').append('<p>Last Update: <strong>' +  displayDate + '</strong></p>');

  });

};
```

7. Manual de Usuario

7.1 Introducción

En este documento se explica cómo se ha de realizar la instalación del software para el SCADA Online y además contiene un manual de instrucciones de uso.

El sistema se puede instalar en plataformas Windows, Mac y Linux; pero este manual está orientado a la instalación en una plataforma Windows.

7.2 Instrucciones de uso

En la página web, que está alojada en la IP *http://155.210.159.208:80/home*, se encuentra un manual básico del funcionamiento de la aplicación web. Por lo tanto, este documento se centra en un manual para un uso más avanzado.

7.2.1 Acceso al menú de administrador

La aplicación web creada por el framework de Django pone a disposición un menú en el navegador web del usuario final donde se pueden editar los *modelos* además de un conjunto de cosas relacionadas con la aplicación, que en este caso son las tareas periódicas.

Para acceder a este menú, a la derecha de la barra de navegación de la página web se encuentra un menú desplegable donde el usuario pulsando el botón *Settings* puede acceder al menú de administrador, como se ve en la Figura 1.

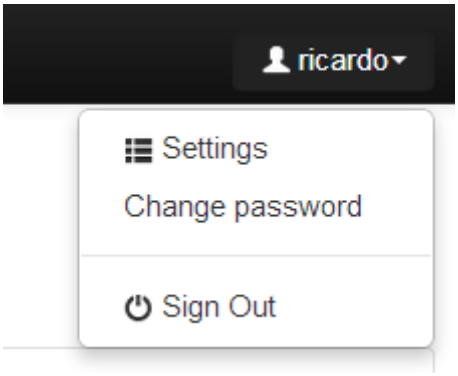


Figura 45 Menú desplegable del usuario.

7.2.3 Adición de variables y tareas periódicas

La inclusión de variables y tareas periódicas nuevas se trata en los apartados *3.5.2* y *3.5.4*, respectivamente, de la Memoria de este proyecto, entonces este apartado solo se centrará en cómo acceder al menú descrito en los apartados explicativos de la memoria, que están dentro del menú de administrador, en los apartados *Variables->Variables* y *Djcelery->Periodic Tasks,* respectivamente y como se puede apreciar en la Figura 2 y Figura 3.

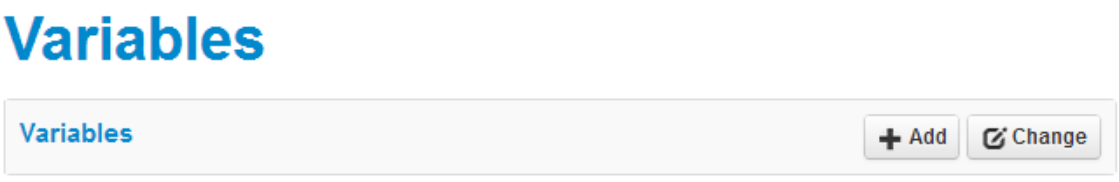


Figura 46 Menú de administrador para *Variables.*

Figura 47 Menú de administrador para Tareas periódicas.

7.2.3 Creación de usuarios del sistema

Al igual que para las variables y las tareas periódicas, existe un menú para el control de usuario del sistema. A través de este menú, el usuario que actúa como *superusuario* puede añadir usuarios nuevos y dotarles de los credenciales para acceder a las distintas tareas del SCADA. De esta manera se puede configurar que es lo qué haría un usuario en el sistema: se puede configurar que pueda entrar al SCADA pero solo para realizar tareas de lectura, otro que pueda realizar operaciones de control en los PLCs pero no añadir usuarios nuevos o también un usuario que pueda tener la funcionalidad completa del SCADA su disposición.

Para gestionar los usuarios se acceder a *Auth->Users* dentro del menú de administrador. Como se ve en la Figura

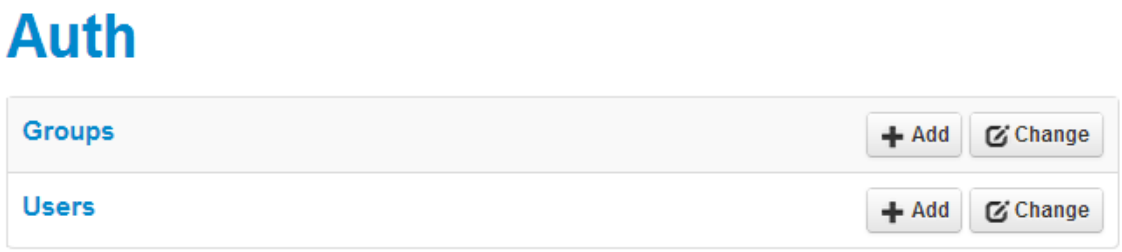


Figura 48 Menú de gestión de usuario del sistema SCADA.

Durante el proceso de creación de un nuevo usuario se pasa por formularios para indicar el nuevo nombre y contraseña, como indica la Figura 5

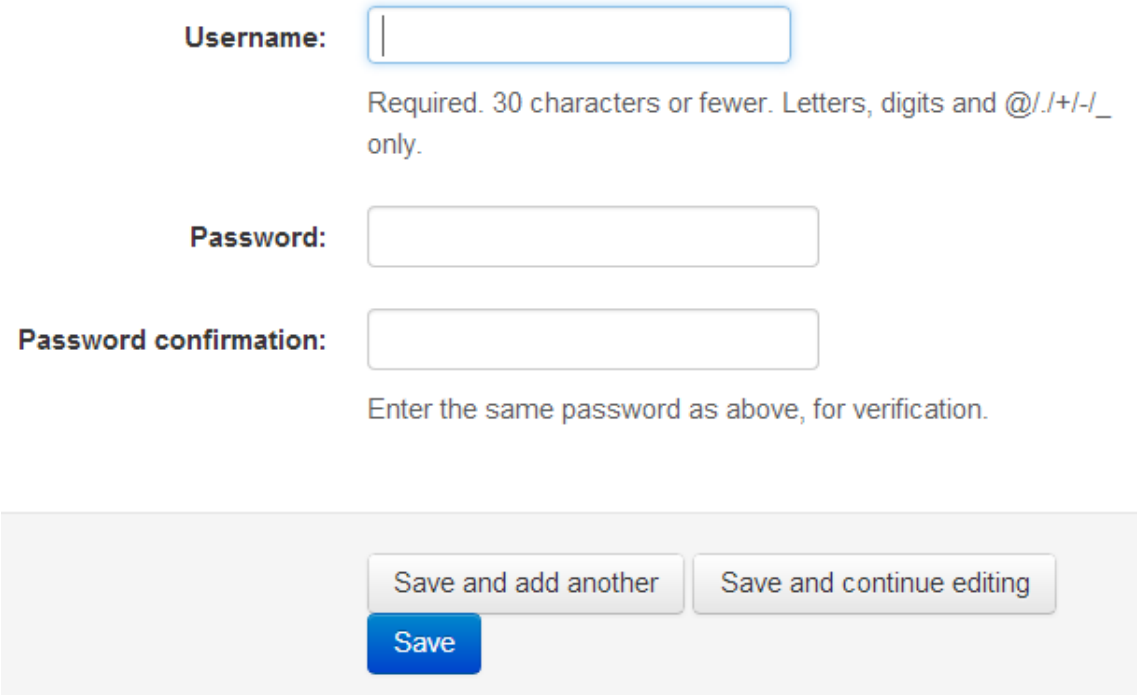


Figura 49 Formulario de creación de un nuevo usuario.

En el siguiente apartado del formulario se indica el nivel de acceso que tendrá el usuario al sistema. En el ejemplo de la Figura 6 se puede observar un usuario con un conjunto de permisos que se pueden configurar para determinar el nivel de acceso deseado para el usuario a crear.

- *Can add user:* para añadir usuarios nuevos al sistema.
- *Can add periodic task:* podrá añadir nuevas tareas periódicas.
- *Can add variable:* nuevas variables se pueden controlar en el sistema.
- *Can change variable:* permite realizar operaciones de escritura en las variables.

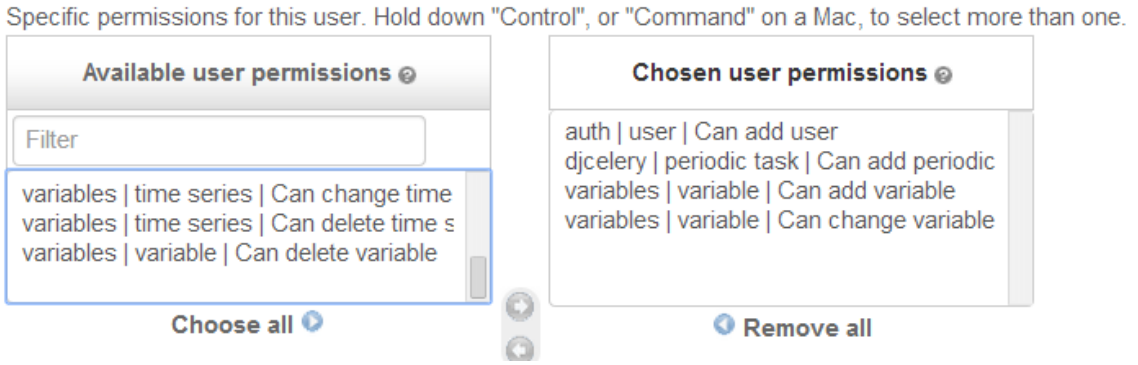


Figura 50 Formulario para los permisos de usuario.

7.3 Instrucciones de instalación

En este apartado se enunciará los paquetes de software a instalar y se explicará cómo se han de incorporar los archivos del autor además de la configuración de los programas pertinentes. Si se está en una plataforma Windows, los paquetes están incluidos en el CD del proyecto.

Software a instalar:

- Python 2.7: Lenguaje de programación usado en el proyecto. Disponible en la web de Python.
- Pip: Herramienta para la instalación y manejo de paquetes Python. Disponible en la web de pip.
- Pywin32: Extensiones de Python para Windows. Disponible en la página web de Pywin320
- OpenOPC: Cliente OPC usado en el SCADA. Disponible en la página web de OpenOPC for Python.
- Apache2.2: Servidor HTTP recomendado para el proyecto. Disponible en la página web de Apache.
- RabbitMQ: Para las tareas automatizadas. Disponible en la web de RabbitMQ.

Adicionalmente a estos paquetes de software, que son archivos ejecutables en la plataforma Windows, se instalarán paquetes python a través de la consola de comandos. Esto se detallará a continuación.

7.4.1 Paquetes Python

Como se ha explicado, python es el lenguaje de programación predominante en la aplicación. Aquí se detallan los paquetes adicionales a instalar. Estos paquetes pueden ser instalados en cualquier plataforma, pero aquí se usará Windows como ejemplo.

Para manejar Python en Windows desde la consola de comandos han de configurarse las variables del entorno donde se añadirán a la variable “Path” la siguiente directiva:

- “;C:\Python27;”

Con esto se consigue accede a la consola de Python desde la consola de commandos.

A través de *Pip* instalaremos los siguientes paquetes de Python con las siguientes directivas:

- “pip install django” Django es el framework de la aplicación web.
- “pip install django-celery” Celery es un ejecutor de tareas.
- “pip install pyro” Librería Django para objetos.

```
C:\>pip install django
Downloading/unpacking django
  Downloading Django-1.6.2.tar.gz (6.6MB): 6.6MB downloaded
  Running setup.py egg_info for package django

    warning: no previously-included files matching '__pycache__' found under dir
ectory '*'
    warning: no previously-included files matching '*.py[co]' found under direct
ory '*'
Installing collected packages: django
  Running setup.py install for django

    warning: no previously-included files matching '__pycache__' found under dir
ectory '*'
    warning: no previously-included files matching '*.py[co]' found under direct
ory '*'
Successfully installed django
Cleaning up...

C:\>
```

Figura 51 Ejemplo del uso de *pip* en la consola de comandos Windows.

7.4.2 Archivo de configuración Apache

A la hora de configurar el servidor Apache, hay que tener en cuenta:

- Módulo “mod_wsgi”
- Configuración de “http.conf”

El primer módulo hay que incluirlo en el directorio de módulos de Apache, y sirve para comunicar al servidor con la aplicación Python del SCADA. El módulo se puede bajar de la web de *mod_wsgi* para la plataforma necesaria.

Para el archivo de configuración *http.conf*, se puede sustituir el que viene por defecto y usar el que se suministra en el CD de instalación. A continuación se indican los cambios que habría que hacer en el archivo *http.conf* de no quererse sustituir completamente:

- *“*ServerRoot “C:/Apache2.2””. O donde esté instalado Apache.
- “LoadModule wsgi_module modules/mod_wsgi.so”. Para indicar que cargamos el módulo wsgi y el directorio donde está alojado.
- “DocumentRoot “C:/Apache2.2/htdocs”.”
- Las siguientes directivas que se recogen en la imagen corresponden a la configuración para el interfaz wsgi. Sirven para definir, donde se encuentra el archivo para comunicar la aplicación web con el servidor Apache y los directorios en los que Apache puede acceder para suministrar archivos al usuario como son los documentos estáticos por ejemplo.

```
WSGIScriptAlias / C:/django_scada/django_scada/wsgi.py
WSGIPythonPath C:/django_scada

Alias /static/ C:/django_scada/apache/static/

#Directiva para suministrar los archivos estaticos
<Directory C:/django_scada/apache/static/>
Order deny,allow
Allow from all
</Directory>



#Directiva para suministrar el archivo wsgi
<Directory C:/django_scada/django_scada/>
<Files wsgi.py>
Order deny,allow
Allow from all
</Files>
</Directory>
```

Figura 52 Directivas aplicables al interfaz wsgi en Apache.