Memoria
Trabajo
Servicios Web
REST

Aplicaciones Distribuidas

Victor Paz Rodríguez DNI 42187180M

vpaz14@alumno.uned.es

Tlfo: 616048996

Arquitectura REST Desarrollada:

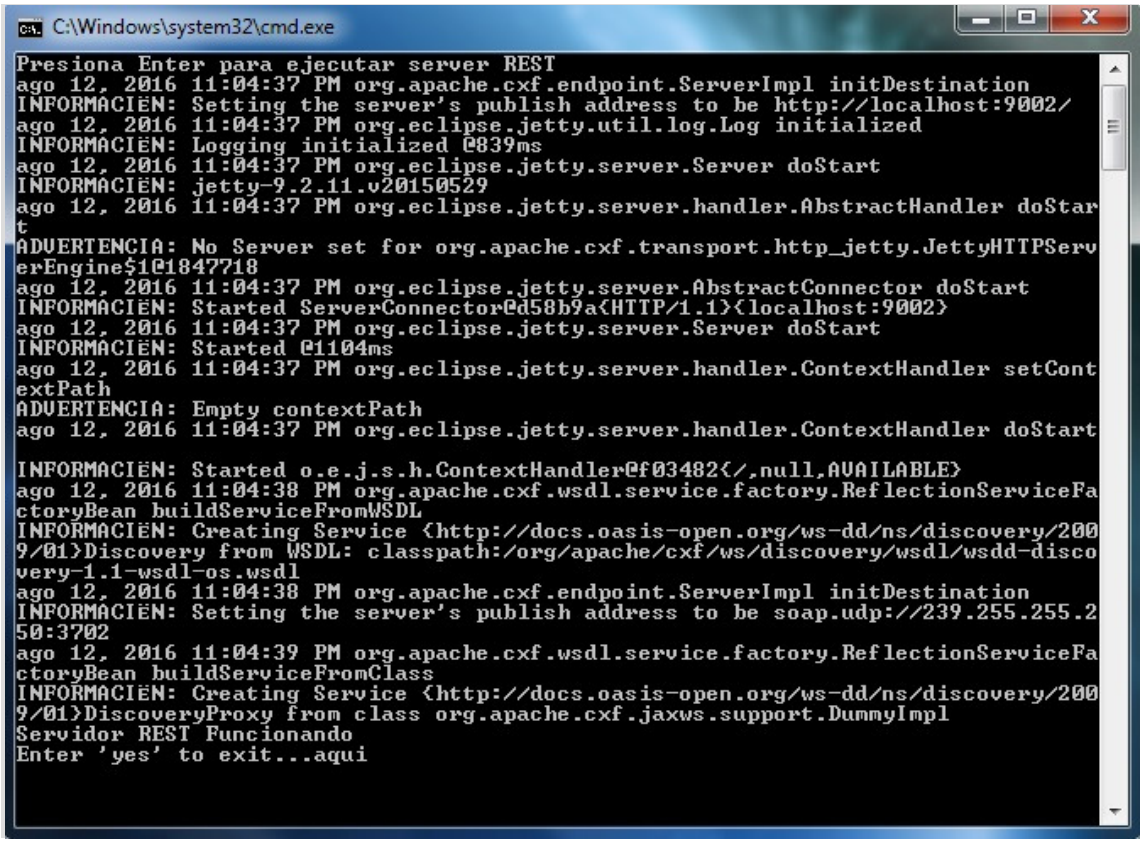
La arquitectura de este trabajo ha sido desarrollada según se indica en el enunciado propuesto por el equipo docente.

En primer lugar he creado la interfaz del servicio REST denominada RESTSignalGenerator copiando el código que se nos proporciona en el enunciado del trabajo, esta interfaz es muy parecida a la del trabajo SOAP, lo único que cambia es la ausencia de las anotaciones que había en la interfaz SOAP.

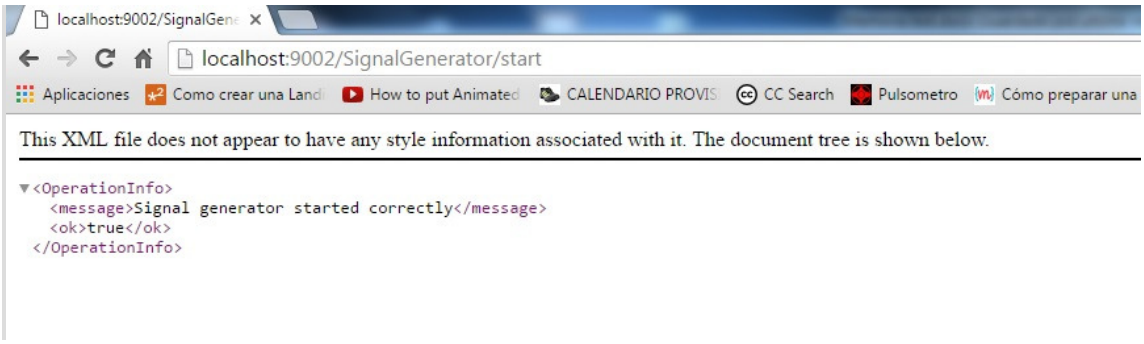
Posteriormente he creado la clase RESTSignalGeneratorWSImpl que implementa los métodos de la interface del servicio como indica el enunciado. La implementación de los métodos me resultó sencilla al ser prácticamente igual que la realizada en el trabajo SOAP, salvo porque delante de cada método introducimos las anotaciones @Path para definir el punto de entrada al servicio, @GET que define una operación Get, normalmente sólo debe usarse cuando queremos leer información, @Produces que indica el formato en el que es generado es contenido del servicio REST y @Post que solo la utiliza el método setSignalParameters ya que se usa para añadir un recurso o modificar un recurso existente . La Clase RESTSignalGeneratorWSImpl implementa los siguientes métodos. **Start** que es el encargado de poner a funcionar el generador de señales, **Stop** encargado de parar el generador de señales, **getSignalValue** que lee los valores de tiempo y los valores de salida del generador, **getSignalParameters** que lee los valores los parámetros (amplitud, frecuencia y tipo de señal) que le pasamos al generador y el método **setSignalParameters** que se encarga de fijar el valor de los distintos parámetros que le pasamos al generador para su funcionamiento. También está implementado el método **isrunning** que comprueba si el generador está en funcionamiento o no.

Una vez implementada la interfaz remota procedí a crear la clase RESTWSServer. Para la implementación de esta clase usé la ayuda proporcionada en el enunciado por el equipo docente. La clase WSServer consta de dos métodos; un método llamado **waitForKey** que es el encargado de mantener el servidor REST funcionando hasta que el usuario decida pararlo**,** es el mismo que proporciona el equipo docente en el enunciado del trabajo RMI. Finalmente hay un método principal que contiene las sentencias que vienen en el enunciado para usar las funcionalidades CFX e invoca al método citado anteriormente. La creación del servidor no me dio problemas, pero al crear el archivo bat del servidor si me creó problemas al referenciar el classpath de las librerías CFX, ya que al poner el path del enunciado ./lib/apache-cxf-3.1.3/lib no me arrancaba correctamente el servidor, que si me arrancaba sin problemas en el eclipse, para solucionarlo y que todo me funcione correctamente he puesto el path **./lib/apache-cxf-3.1.3/*.**

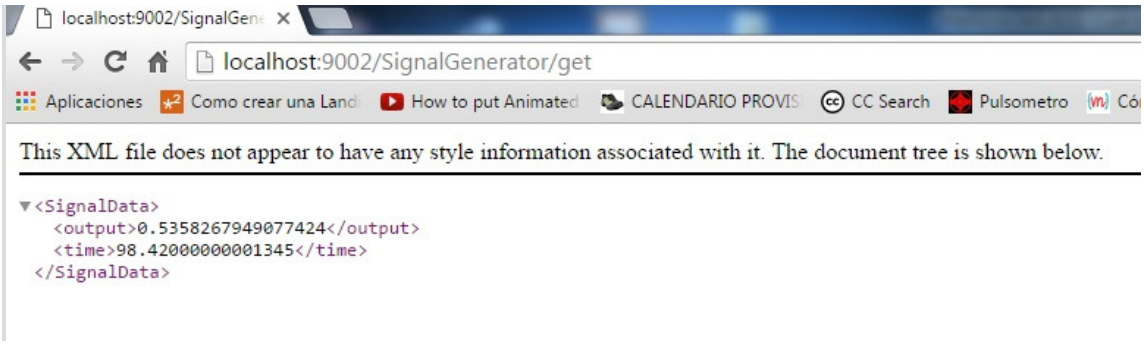
A continuación se muestra una captura de pantalla del servidor REST funcionando.



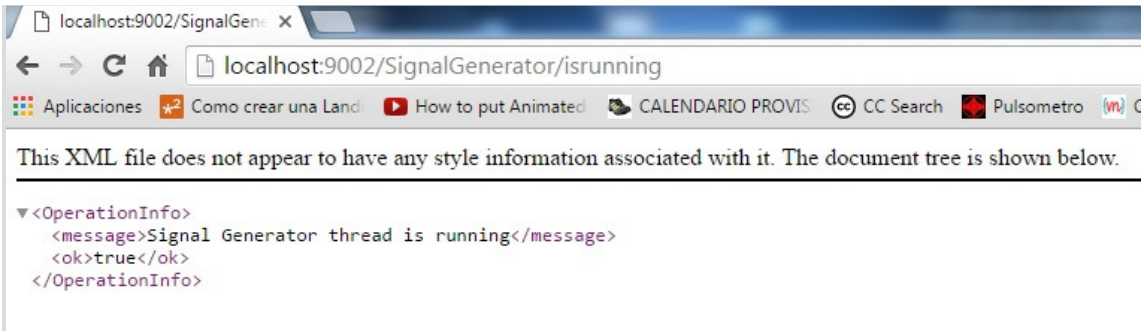
Una ver ejecutado el servidor REST procedo a invocar las operaciones del servicio con un cliente HTTP.
En la siguiente imagen se puede ver una captura de la invocación del servicio start para arrancar el generador de señales. La URI intruducida es: http://localhost:9002/SignalGenerator/start



En la siguiente imagen se puede ver una captura de la invocación del servicio get para obtener el valor de la señal. La URI introducida es: http://localhost:9002/SignalGenerator/get

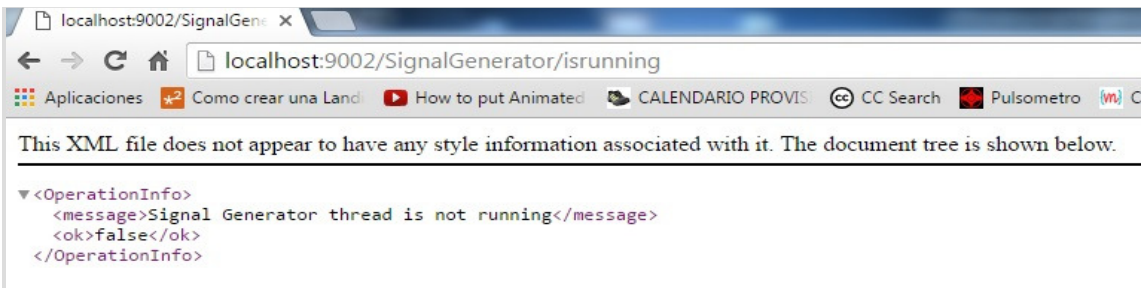


En la siguiente imagen se puede ver una captura de la invocación del servicio isrunning para comprobar que el generador de señales está ejecutándose. La URI introducida es: http://localhost:9002/SignalGenerator/isrunning



En la siguiente imagen se puede ver una captura de la invocación del servicio stop para detener el generador de señales. La URI introducida es: http://localhost:9002/SignalGenerator/stop

En la siguiente imagen se puede ver una captura de la invocación del servicio isrunning para comprobar el estado del generador de señales. La URI introducida es: http://localhost:9002/SignalGenerator/isrunning



Posteriormente procedí implementar la clase RESTClient, esta clase sólo consta de un método principal, este método simplemente crea un objeto de la clase webclient en el punto de publicación del servicio, este objeto se le pasa como parámetro a la instancia que creamos de la clase auxiliar PlottingFrame que describiremos a continuación.

A continuación se muestra una captura de pantalla del cliente REST funcionando.

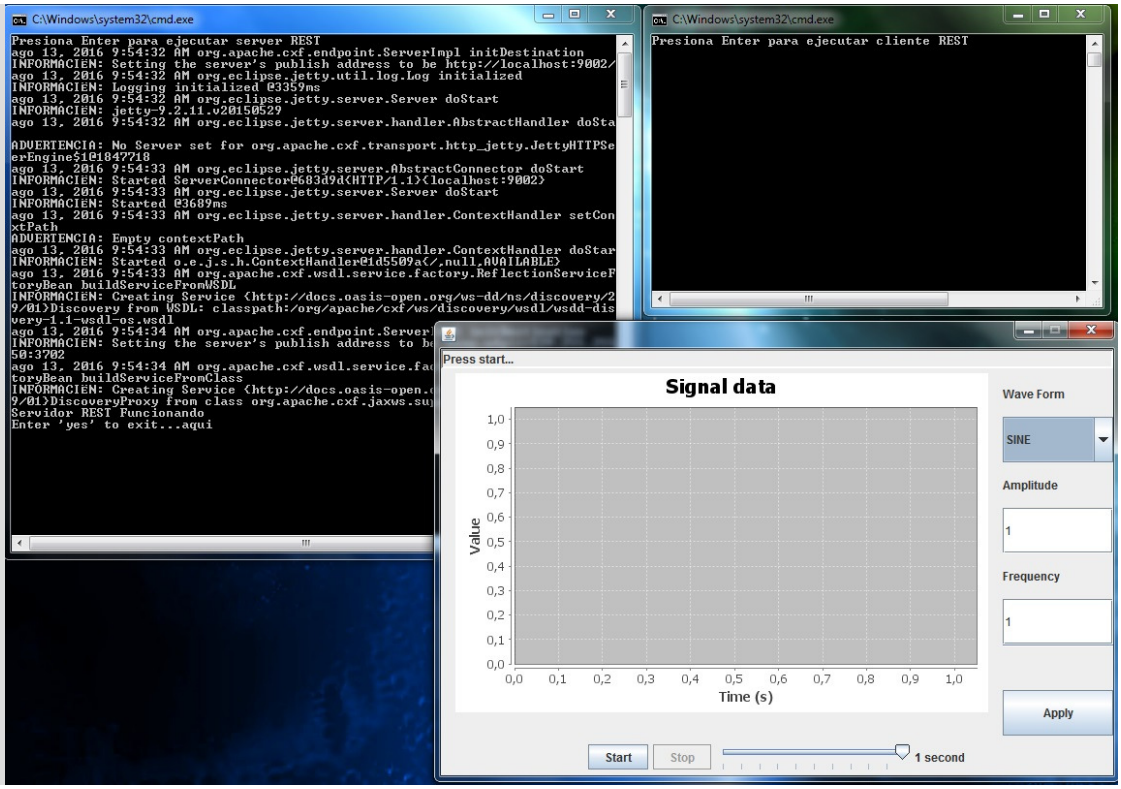
Finalmente implementé la clase PlottingFrame, esta clase implementa la interface ClientPlot suministrada por el equipo docente. En el constructor de la clase le asignamos al objeto restClient la referencia del servicio que hemos pasado por parámetro al instanciar la clase.

A continuación implemente las funciones de la interface siguiendo el ejemplo del enunciado. En el método start (copiado del ejemplo del enunciado) primero establecemos la ruta relativa al método, luego establecemos el tipo de datos recibidos, posteriormente realizamos la llamada al método y para finalizar obtenemos la entidad XML de respuesta. El método start que hace que cuando se pulsa el botón start de la interface gráfica se invoque al servicio que pone a funcionar el generador, el método stop que es análogo al start y se implementa igual, pero lo que hace es parar el generador.El método getSignalValue que es el encargado de que se muestren los distintos valores de salida y de tiempo en la pantalla se implementa de forma análoga la método start, el método getSignalParameters que nos muestra los parámetros con los que está funcionando el generador (inicialmente la frecuencia y la amplitud se inicializan a 1 y el tipo de señal a sine), se implementa de forma análoga a los métodos anteriores , y el método setSignalParameters que es el encargado de fijar los valores de los parámetros, al principio con los valores iniciales y luego cuando pulsamos el botón apply para aplicar los cambios realizados a los parámetros. Este último método sólo se diferencia en los anteriores en que usa la anotación post en vez de get por lo motivos que hemos expuesto anteriormente, pero la implementación es análoga a los métodos anteriores.

Todos estos métodos hacen uso del servicio REST y de la implementación de estos que hemos realizado en la clase RESTSignalGeneratorWSImpl

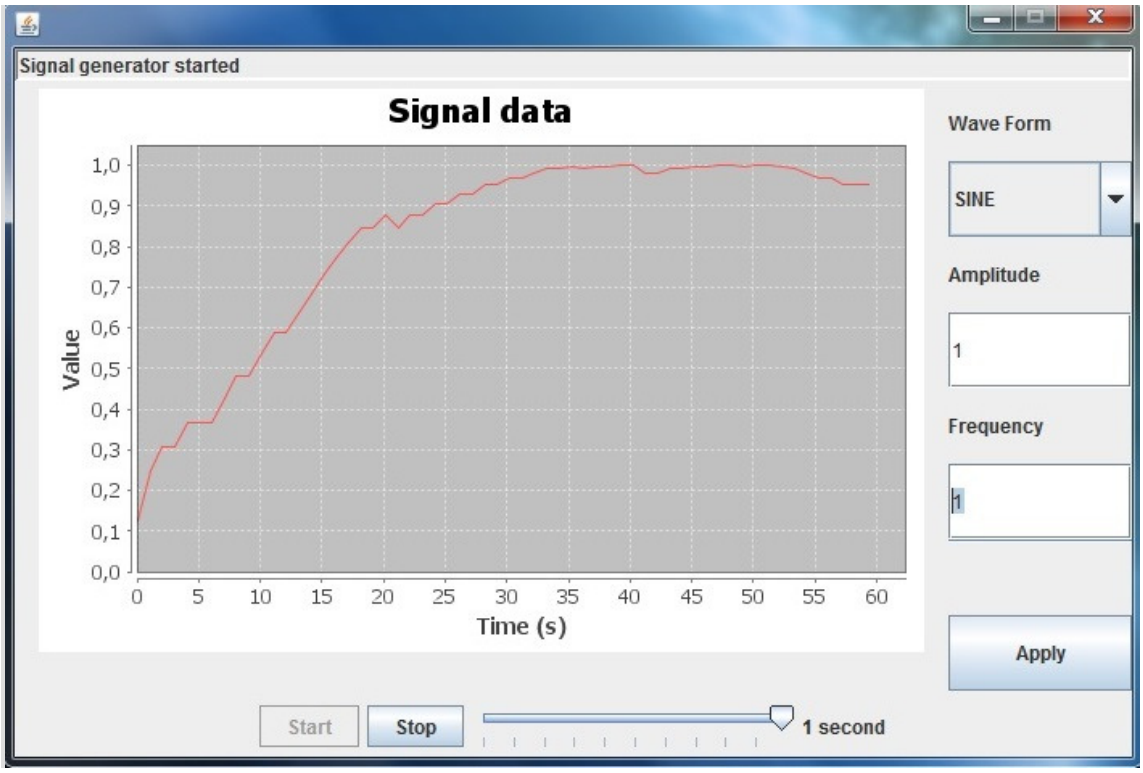
Pruebas REST:

En la imagen que se muestra a continuación se puede observar la interface gráfica que aparece después de ejecutar el servidor y el cliente REST, tal como se pide en el enunciado.

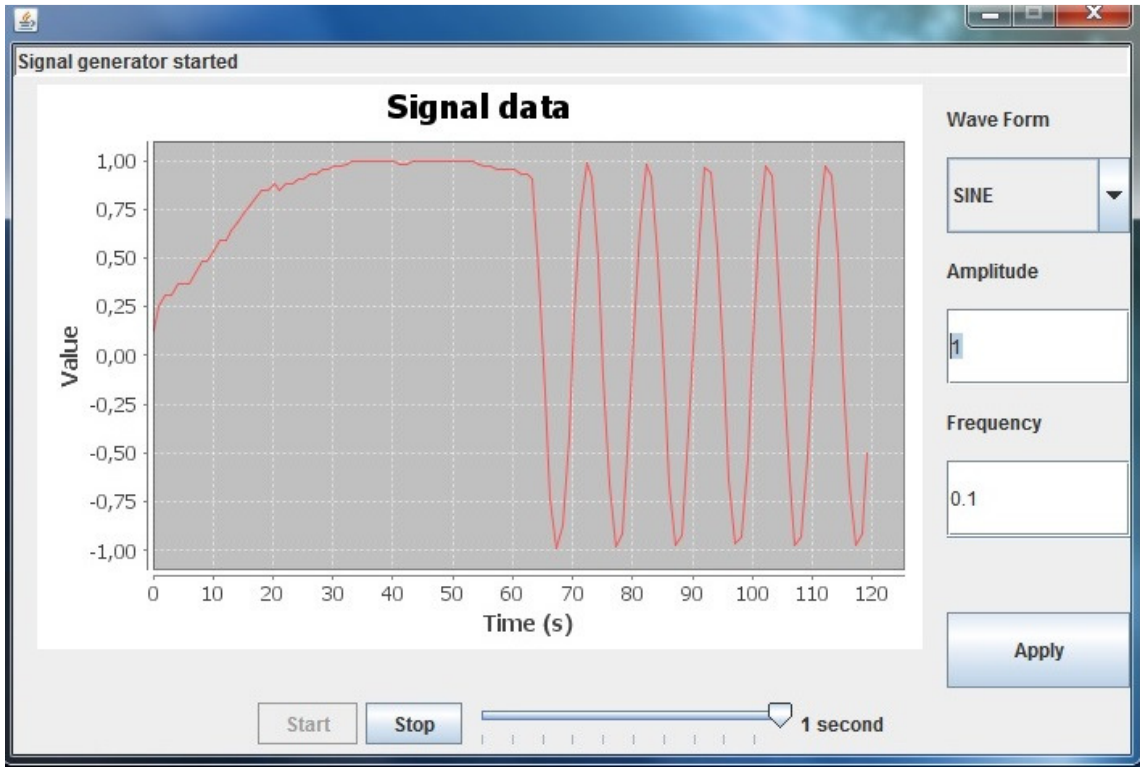


En la siguiente imagen podemos observar la GUI después de un minuto. Se puede observar que la imagen con los parámetros iniciales (Amplitud=1, Frecuencia=1, Señal=sine) dibuja la curva ascendente de un seno desde 0 hasta 1. Yo diría que no presenta anomalías, puede haya alguna entorno al segundo 20 que puede ser debida a algún posible fallo en la comunicación o como me parece más probable a algún problema en el proceso serialización o recomposición de los parámetros, pero la curva parece bastante clara y no es nada importante.

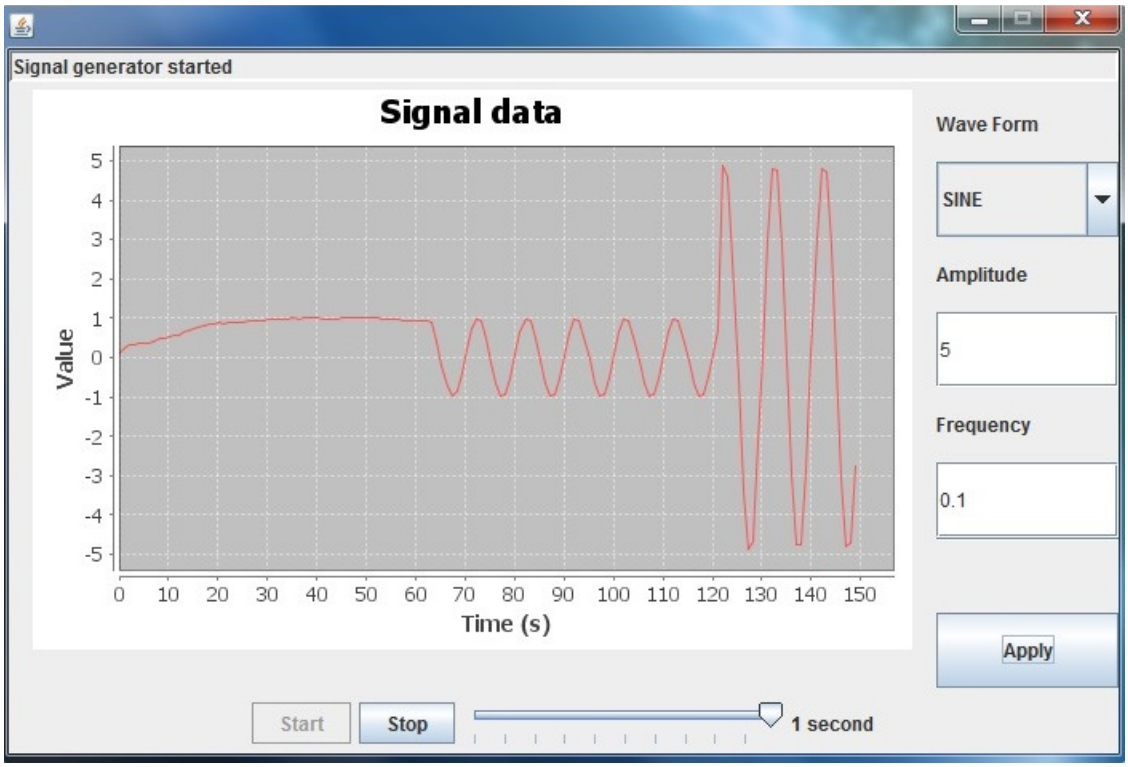
Aunque el comportamiento habitual de la señal es el que se muestra en las siguientes imágenes, hay ocasiones en que la señal presenta un comportamiento totalmente aleatorio desde que comienza su ejecución. No sé a que se debe este comportamiento extraño de la señal en algunas de las ejecuciones del servicio.



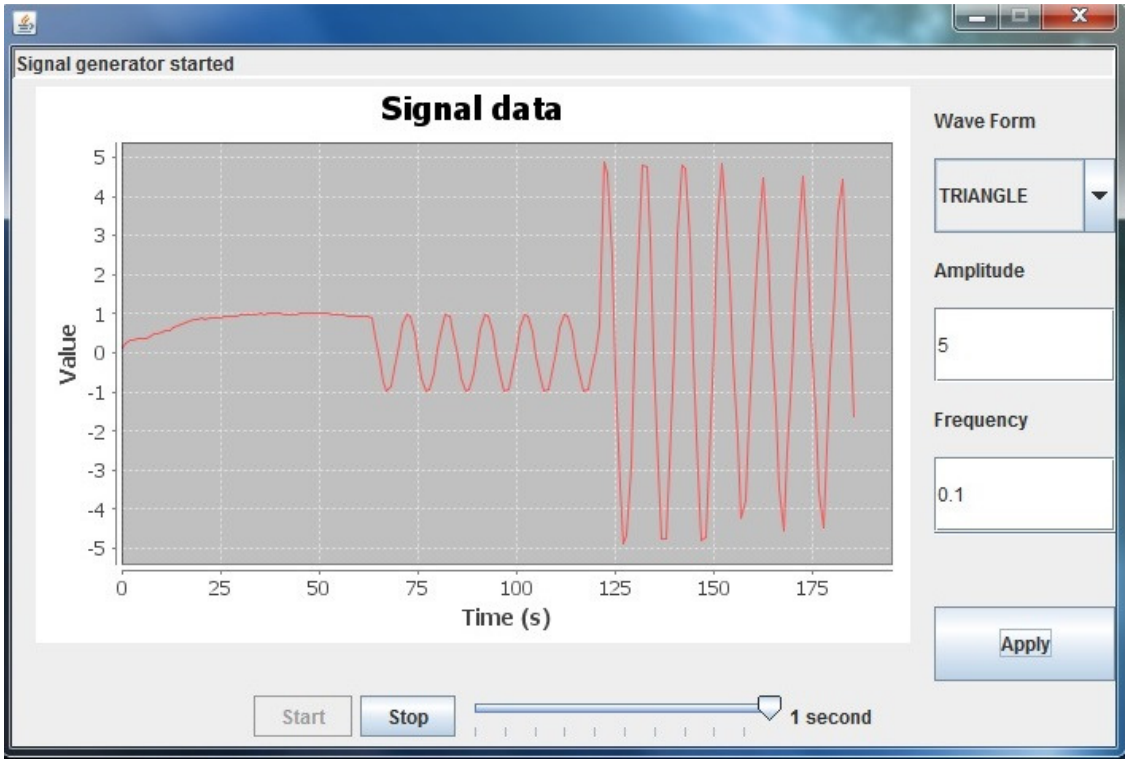
En la imagen que se muestra a continuación podemos ver la imagen después de dos minutos con una frecuencia de 0.1 después del primer minuto. En esta imagen se observa ya claramente la señal sinusoidal al ser la frecuencia mucho más pequeña.



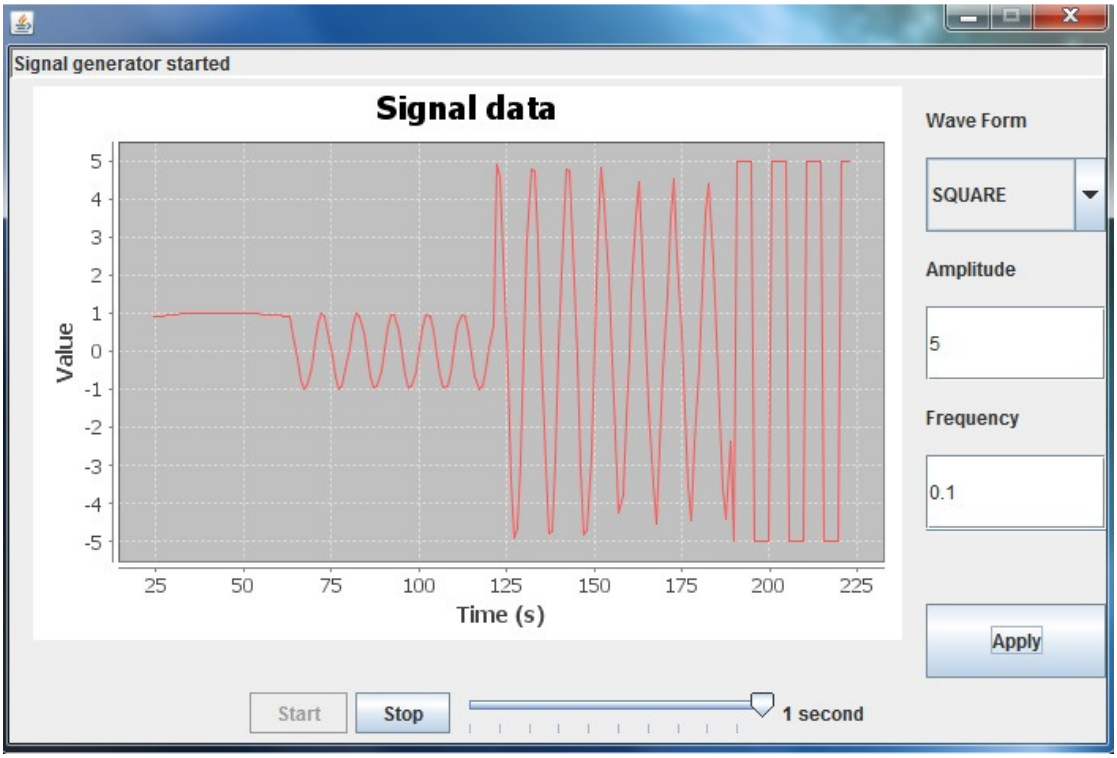
En la siguente imagen podemos observar la señal despues de dos minutos y medio en la que vemos claramente como despues de dos minutos hemos aumentado la amplitud de la señal sinusoidal a 5.



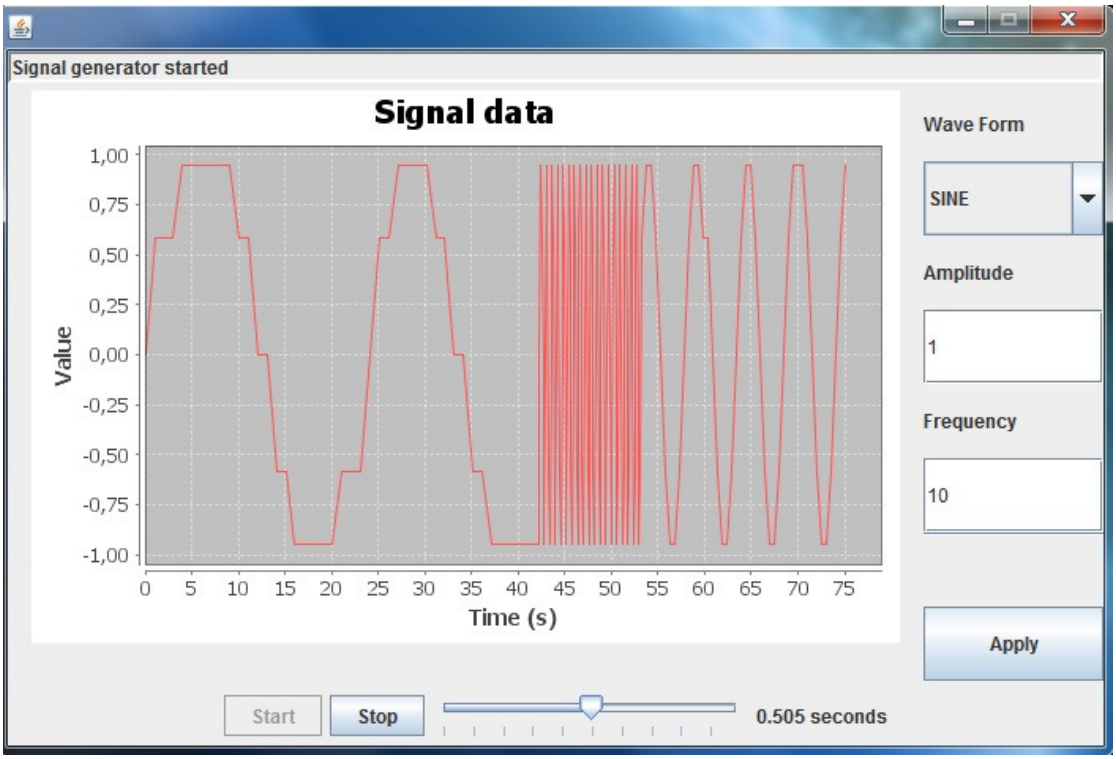
En la siguente imagen podemos observar la señal despues de tres minutos en la que vemos claramente como después de dos minutos minutos y medio la forma de la señal cambia y se vuelve triangular.



A continuación vemos la imagen después de tres minutos y medio, donde se puede observar claramente que la señal ha pasado a ser cuadrada



En esta última imagen podemos ver el efecto del slider cuando vamos cambiando los valores de este. Vemos que al principio de la gráfica cuando el slider tiene un valor de 1s, da la sensación de que los puntos están más espaciados y la velocidad a la que se cogen los puntos del generador y se muestran en pantalla es mucho más lenta que en la siguiente parte de la gráfica cuando ponemos el slider a 0.1, aquí los puntos están mucho más próximos y la velocidad a la que se muestra la gráfica en pantalla es mucho mayor. Por último cuando ponemos el slider a 0.5 nos encontramos en un punto intermedio de lo comentado anteriormente.


Signal generator started
Signal data
1,00
0,75
0,50
0,25
0,00
-0,25
-0,50
-0,75
-1,00
Value
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75
Time (s)
Wave Form
SINE
Amplitude
1
Frequency
10
Apply
Start
Stop
0.505 seconds