

easySP: Nueva Aplicación Para la Enseñanza de Procesado de Señal

Javier Vicente, Begoña García, *Member, IEEE*, Ibon Ruiz, Amaia Méndez, Oscar Lage

Abstract— The present article presents a software application, developed by the authors, in the digital signal processing field, with educative purposes. This tool offers a graphic interface to perform signal processing presentations via plugins. The program allows users to add plugins designed in both XML format and Java. Both kind of plugins allow the incorporation of Octave/Matlab functions in order to implement signal processing algorithms. In one hand, this application allows the student to interact with presentations changing their parameters and, in the other hand, offers the possibility of implementing new signal processing plugins easily

Index Terms—Education, Signal processing, Software prototyping, Simulation software

I. INTRODUCCIÓN

Tras años de experiencia de los autores del artículo en la enseñanza del procesado digital de señal [1][2] se llegó a la siguiente conclusión: los alumnos asimilan mejor los conocimientos si disponen de demostraciones visuales con las que poder interactuar. Siguiendo esta premisa, se comenzaron a desarrollar ejemplos interactivos con los que se pudiese analizar las distintas áreas del procesado de señal. Por ejemplo, si se quiere explicar a los estudiantes el funcionamiento de un filtro paso-bajo Chebychev, con esta herramienta se les entrega una aplicación en la que los alumnos pueden variar los parámetros que definen el filtro, y comprobar como afectan éstos al diagrama de modulo y fase de la respuesta frecuencial. No obstante, aunque estas aplicaciones permiten una comprensión más efectiva de las técnicas de procesado de señal, también se pretende que los futuros ingenieros alcancen un nivel de conocimiento superior de las técnicas de procesado de señal. Para que los estudiantes llegasen a este nivel, hasta ahora, se les planteaban prácticas en las que ellos mismos tenían que programar algoritmos de procesado de señal. Hasta el momento, estas prácticas se realizaban en Octave [3][4], pero tenían el inconveniente que con Octave no se podían realizar interfaces gráficas de usuario. Este hecho, imposibilitaba que los alumnos pudiesen lograr aplicaciones interactivas semejantes a las que se les entregaban a ellos.

Los autores pertenecen al departamento de Arquitectura de Computadores, Automática y Electrónica, y Telecomunicaciones de la facultad de ingeniería (ESIDE) de la Universidad de Deusto (e-mail: jvicente@eside.deusto.es).

DOI (Digital Object Identifier) Pendiente.

El primer paso para posibilitar que los alumnos pudiesen dotar de un interfaz de usuario a sus algoritmos realizados en Octave fue la API joPAS [10]. Esta API permite implementar en Java el GUI de una forma rápida, manteniendo los cálculos de procesado de señal en Octave.

En una segunda aproximación, surgió easySP, una aplicación que permite dotar a funciones de Octave de GUIs de estructura predefinida, sólo con definir un sencillo fichero XML en el que se especifican los elementos que debe poseer el interfaz de usuario. Permitiendo de esta manera, generar demostraciones de algoritmos de procesado de señal de una forma muy sencilla y rápida, sin tener que dominar otro lenguaje adicional, sino únicamente Octave, en los casos sencillos [5][6].

Otra posibilidad, es mediante la creación de plugins en Java, estos deben tener una estructura definida para que easySP los pueda detectar y añadir a su menú. Este tipo de plugins requiere un mayor esfuerzo de desarrollo, pero proporcionan una mayor versatilidad a la hora de diseñar el interfaz de usuario.

A continuación, se describen los principales objetivos planteados por los autores en el comienzo del desarrollo de esta experiencia, los métodos utilizados, y los resultados alcanzados tanto a nivel técnico como educativo.

II. OBJETIVOS

El principal objetivo planteado es obtener una herramienta que permita el estudio del procesado de señal de una forma eficaz, rápida y amena, para conseguir aumentar la motivación y satisfacción de los estudiantes. Dividiendo este aprendizaje en dos niveles de profundidad:

- En un primer nivel, el estudiante es un simple usuario de la herramienta, e interactúa con las demostraciones que posea de cada área del procesado de señal.
- En un segundo nivel, el alumno debe tener una actitud más creativa, siendo él mismo, el que tenga que incrementar las opciones de la herramienta añadiendo nuevas demostraciones de procesado de señal.

Para que la aplicación sea realmente efectiva tiene que cumplir los siguientes objetivos funcionales:

- *Basado en software libre* → De este modo, se consigue una aplicación que se distribuirá libremente. Los alumnos pueden instalarla en sus ordenadores personales sin tener que pagar ningún tipo de licencia.
- *Multiplataforma* → La misma aplicación funciona en distintos sistemas operativos sin tener que realizar excesivos cambios.
- *Intuitivo* → El programa posee un interfaz muy intuitivo que permite al alumno interactuar con la herramienta muy rápidamente.
- *Completo* → La herramienta debe disponer de ejemplos que abarquen todas las áreas del procesado digital de señal. Estos ejemplos están divididos en categorías, que son las siguientes: Diseño de filtros, modulaciones y análisis frecuencial de señales.
- *De código abierto* → Dar la posibilidad a los alumnos para que puedan acceder a las instrucciones que se realizan para implementar cada uno de los ejemplos que compone la aplicación.
- *Escalable* → La aplicación tiene que ser muy fácilmente ampliable mediante la incorporación de sencillos plugins. Estos plugins permiten que los propios alumnos aumenten las opciones de la herramienta, aportando ellos mismos nuevos ejemplos. La generación de los plugins ha de ser sencilla, sin necesidad de poseer grandes conocimientos en lenguajes que no estén orientados a la implementación de algoritmos de procesado de señal.
- *Actualizable* → La herramienta es capaz de detectar si existen nuevos ejemplos disponibles, y descargarlos automáticamente para aumentar las opciones de la aplicación.

III. MÉTODOS

En este punto, se describen los métodos utilizados para el desarrollo de la herramienta easySP. Para ello, en la tabla I se destacan las diferentes alternativas existentes, y cuál ha sido la elección final de los autores.

TABLA I. Comparativa de tecnologías

	Elección	Alternativa
Lenguaje de programación	Java	C/C++
Lenguaje científico	Octave	Matlab
Interfaz gráfico	joPAS/XML	Octave-GTK

A continuación, se detallan las características principales de las tecnologías empleadas y la justificación de su elección frente a las diferentes alternativas.

A. Octave

Octave es un lenguaje de alto nivel para el cálculo numérico, siendo su sintaxis compatible con Matlab. Octave es ampliamente utilizado por el entorno docente, y

desarrollado por la comunidad de software libre [7][8]. Octave al ser un proyecto Open Source no tiene la limitación de Matlab, es decir, tener que pagar una licencia para poder ejecutar aplicaciones desarrolladas mediante este entorno.

Octave es un lenguaje especialmente orientado al mundo científico [9]. Entre sus principales diferencias con otros lenguajes de programación destacan las siguientes:

- Operación con matrices de forma nativa.
- Operación con números complejos de forma nativa.
- Lenguaje interpretado.

Estas características permiten que los algoritmos científicos se desarrollen en mucho menos tiempo que en otros lenguajes de programación. De modo que Octave es el lenguaje ideal para el desarrollo de algoritmos de procesado digital de la señal, procesado digital de imagen, sistemas de control, estadística...

Además, existen gran cantidad de toolboxes, que permiten que no se tenga que comenzar desde cero cuando se quiera abordar una temática. Por ejemplo, si alguien quiere desarrollar un algoritmo de procesado digital de voz y necesita filtrar la señal mediante un filtro Butterworth, no necesita implementar esta funcionalidad ya que ésta existe en el toolbox de procesado de señal, de modo que en su algoritmo no tendría más que hacer uso de la misma.

Este tipo de toolboxes tan especializados en temas científicos, no suelen existir en otros lenguajes de programación, de modo que es una ventaja más para desarrollar este tipo de aplicaciones en Octave.

B. joPAS

La API joPAS [10] desarrollada por el grupo PAS de la Universidad de Deusto permite el uso de la potencia de cálculo de Octave desde una aplicación Java.

Existe un proyecto llamado Octave-GTK [11] que proporciona unas características similares a la API joPAS, pero no se ajustaba adecuadamente a los objetivos planteados. Este proyecto, trata de crear un envoltorio GTK (GIMP toolkit) a Octave. Esto permite a los usuarios desarrollar programas con interfaz de usuario implementados en GTK con la potencia de cálculo de Octave.

La diferencia principal de ambos proyectos es el lenguaje mediante el que se implementa el envoltorio de Octave, mientras en Octave-GTK se emplea C, en joPAS se utiliza Java, que es el lenguaje de programación en el que nuestros alumnos están habituados a trabajar.

Mediante joPAS el usuario puede programar aplicaciones en Java [12][13][14], delegando todos los cálculos matemáticos en Octave. Así, joPAS es utilizado como nexo de unión entre Java y Octave. Mediante joPAS se pueden convertir variables de Java en variables Octave, ejecutar funciones de octave y convertir variables de Octave en variables Java. Las clases más importantes de esta API son las siguientes:

- *Jopas* → Esta es la clase fundamental de la API, es la encargada de gestionar la comunicación con Octave mediante tres métodos: load, save y execute.
- *Matrix* → Esta clase contiene el tipo de datos que entiende la clase jopas. Esta clase es un contenedor de matrices, que pueden ser reales o complejas. La clase jopas admite en el método load ese tipo de datos y en el método save siempre devuelve un objeto tipo Matriz.
- *JopasLabel* → Esta clase es una reimplementación de la clase JLabel de Java, que mediante el método “print” permite realizar una representación gráfica de una señal.

A continuación, se puede observar un breve fragmento de código en el que se hace uso de las clases comentadas.

```
Double a = 2;
Matrix mA= new Matrix (a,"a");
Jopas.Load(mA);
jopas.execute("b=a+4");
Matrix mB = jopas.Save("b");
```

Figura 1. Código de ejemplo de uso de joPAS.

En el fragmento de la figura 1 se puede observar como se crea un objeto matriz que contiene un escalar, esta matriz se carga en Octave con el método Load. Después se ejecuta una sentencia de Octave, que suma un escalar a la variable generada. Para finalizar, se recupera el resultado obtenido en Octave, generando un objeto matriz en Java.

La estructura de la API se ha simplificado de tal forma que prácticamente solo hace falta saber utilizar tres clases de la misma para poder desarrollar aplicaciones con un interfaz gráfico. Todo el proceso de comunicación entre Java y Octave, cargar variables de Java a Octave, ejecutar sentencias de Octave y salvar variables de Octave a Java, las realiza la API de forma transparente al usuario. Por lo que el tiempo que se tarda en aprender a utilizar joPAS es mínimo, para alumnos que sepan programar en Octave y en Java, como es nuestro caso.

C. XML

XML es un lenguaje de marcas extensible desarrollado por el World Wide Web Consortium (W3C) [15]. XML es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos. A pesar de que XML se esté utilizando en mayor medida para Internet, se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas.

La principal ventaja que ofrece del uso de ficheros XML es la facilidad de leer y editar dichos archivos, cualquier usuario puede editar un archivo XML desde cualquier procesador de textos que sea capaz de producir archivos de texto plano. XML es un lenguaje muy sencillo de leer, interpretar y modificar. Además desde las herramientas de desarrollo se

facilita su lectura y modificación, ya que estas herramientas reconocen los formatos y ayudan a generar ficheros XML bien formados.

Por todo lo anterior, se ha elegido la tecnología XML a la hora de definir la parametrización de una modulación o procesado de la señal básica, así el estudiante no dependerá de ninguna herramienta en concreto para su edición, y podrá validar si el documento que ha generado está bien formado simplemente abriéndolo con cualquier navegador Web.

La tecnología XML ha permitido crear un lenguaje sencillo para la configuración de los procesados. El usuario definirá los parámetros de entrada, las gráficas que la aplicación debe sacar, así como las funciones que deberá utilizar la aplicación para procesar la señal de entrada.

IV. DISEÑO

En este apartado, se describe cómo se pueden diseñar nuevas extensiones basadas en los métodos anteriormente descritos, para que la aplicación easySP las pueda reconocer e incorporar, tal como se puede observar en la figura 4.

Se ha pretendido crear una aplicación intuitiva para el alumno pero a la vez dotada de la potencia de las aplicaciones modulares. Así, el estudiante/profesor puede probar los plugins ya desarrollados que se entregan con la aplicación para una mejor comprensión de todos los sistemas de procesado de señal básicos. Pero, a la vez se permite ampliar la aplicación creando sencillos plugins. Estos pueden ser desarrollados tanto por profesores que quieran entregar ejemplos de procesado de señal no incluidos entre los ya entregados con la aplicación, como por alumnos para ampliar sus conocimientos mediante ejercicios o proyectos.

Existen dos formas de implementación de plugins:

- Mediante la definición de un XML-Plugin, que permiten realizar ejercicios de una forma muy sencilla y rápida a través de un fichero de configuración XML. La limitación de este método está en que la interfaz generada por la aplicación tiene una estructura predefinida y cerrada, el usuario tan sólo puede especificar el número de componentes que se visualizarán en cada área.
- Mediante el diseño de joPAS-Plugins. Estos plugins son más elaborados en lo que se refiere al interfaz y la funcionalidad, al ser directamente implementados en Java. Permiten usar cualquier función de este lenguaje y pueden llegar a ser más sofisticados que los anteriores pero ello implica una dificultad extra.

A. XML-Plugin

En este tipo de plugins definen tanto la interfaz gráfica de usuario como el algoritmo de la simulación. Así pues, mediante el uso del XML se define qué contenido albergará cada una de las seis áreas de la interfaz. Las seis áreas en las que se ha dividido la interfaz son los siguientes:

1. Título del plugin.
2. Área de representaciones gráficas.
3. Selección del conjunto de gráficas a visualizar.
4. Área de parámetros de entrada.
5. Botones para la ejecución del procesado.
6. Área de texto para la explicación teórica del algoritmo.

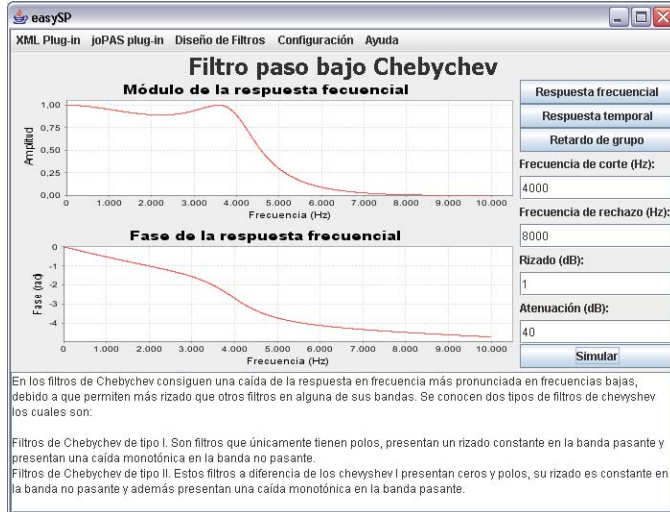


Figura 2. Áreas parametrizables de la interfaz gráfica

Para la definición de la interfaz gráfica se ha definido un lenguaje XML específico. Las etiquetas definidas sirven tanto para indicar qué elementos aparecerán en cada área como para especificar el algoritmo a ejecutar.

Las etiquetas definidas en este lenguaje son las siguientes:

- *Category*: elemento que indica a la aplicación la categoría en la que se listará el procesado. Gracias a este elemento la aplicación ordenará todos los *plugin* que se añadan por categorías.
- *Title*: este elemento contiene el título de la ventana generada por la aplicación.
- *Description*: explicación teórica del procesado o procesados implementados en el plugin.
- *Input*: estos elementos definirán los parámetros de entrada. Permiten definir el valor por defecto que tomará la entrada, el nombre de la variable en Octave y el nombre del parámetro de entrada.
- *Button*: elementos que definirán cada conjunto de gráficas a visualizar.
- *Function*: elemento que define la llamada a la función, o sentencias, de Octave que implemente el procesado de señal.

Para examinar la estructura del fichero de configuración XML se analizará el ejemplo de la figura 3, en el que se ha definido un algoritmo de un filtro paso bajo de Chebychev.

Para la generación del fichero XML de configuración del plugin, se deben identificar los elementos deseados para cada una de las áreas. Una vez realizado se puede comenzar la descripción del contenido del fichero XML.

Esta descripción de los elementos podría ser agrupada por cada una de las áreas, como se puede observar en la figura 2.

```

<?xml version="1.0" encoding="UTF-8"?>
<Plugin version="1.0">
  <Category>Diseño de filtros</Category>
  <Title>Filtro paso bajo Chebychev</Title>
  <Description> Los filtros de Chebychev consiguen una caída de la
  respuesta en frecuencia más pronunciada en frecuencias bajas.
  </Description >
  <Input value="3000" OctaveName="fc">Frecuencia de corte
  (Hz)</Input>
  <Input value="4000" OctaveName="fs">Frecuencia de rechazo
  (Hz)</Input>
  <Input value="1" OctaveName="R">Rizado (dB)</Input>
  <Input value="40" OctaveName="A">Atenuación (dB)</Input>
  <Button>
    <Name>Respuesta Frecuencial</Name>
    <Graph Title="Módulo de la respuesta frecuencial"
      xTitle="Frecuencia (Hz)" yTitle="Amplitud" varX="F"
      varY="m"/>
    <Graph Title="Fase de la respuesta frecuencial" xTitle="Frecuencia
  (Hz)" yTitle="Amplitud" varX="F" varY="p"/>
  </Button>
  <Button>
    <Name>Respuesta temporal</Name>
    <Graph Title="Respuesta impulsional" xTitle="Tiempo (s)"
      yTitle="Amplitud" varX="t" varY="i"/>
    <Graph Title="Respuesta al escalon" xTitle="Tiempo (s)"
      yTitle="Amplitud" varX="t" varY="s"/>
  </Button>
  <Button>
    <Name>Retardo de grupo</Name>
    <Graph Title="Retardo de grupo" xTitle="Frecuencia (Hz)"
      yTitle="Retardo (s)" varX="F" varY="g"/>
  </Button>
  <Function>
    <Name>Simulación</Name>
    <OctaveFile>filterDesign.m</OctaveFile>
    <Callback>[m,p,F,s,i,g,t]=filterDesign(fs,fp,R,A)</Callback>
    <RunOnStartup>true</RunOnStartup>
  </Function>
</Plugin>

```

Figura 3. Código XML del fichero de configuración de ejemplo.

La metodología indicada para realizar dicha descripción es la siguiente:

- Área de Título (área 1): para configurar esta área se configura la etiqueta *Title* indicando el título del plugin.
- Área de Gráficas (área 2): se visualizaran las gráficas definidas en los elementos *Button*, cuando estos sean ejecutados.
- Área de Selección de Gráficas (área 3): mediante los elementos *Button* se define qué señales se quiere representar en cada una de las gráficas anteriores.
- Área de Entrada de Datos (área 4): en esta área se definirían tantos elementos *Input* como parámetros de entrada se deseen; indicando el nombre, valor por defecto y el nombre de la variable en Octave a la que se asociará el valor que contengan.
- Área de Simulación (área 5): en esta área se definirán elementos *Function*, en los cuales se especificarán las sentencias o funciones de Octave que se deseen

ejecutar. Las variables de entrada utilizadas para dicha ejecución tienen que ser las asociadas a los elementos *Input* del área 4, y las variables de salida tienen que ser las utilizadas por los elementos *Button* del área 3.

- Área de Contenidos Teóricos (área 6): mediante el elemento *Description* se definen los contenidos teóricos que se desea que aparezcan en esta área.

Para finalizar con la metodología de creación de ficheros XML se definiría el elemento *Category* indicando en él la categoría del procesado de señal del plugin para su posterior agrupación por parte de la aplicación.



Figura 4. Diagrama de metodología de definición del fichero de configuración.

B. joPAS-Plugin

La implementación de plugins utilizando la API joPAS es mucho más potente, sobre todo en la creación de interfaces gráficas. El usuario, en este caso, sustituye la creación de un fichero XML de configuración por la creación de un archivo .java. Esto significa que el usuario tiene total libertad para desarrollar en Java su plugin. Así el alumno puede desarrollar su creatividad con libertad, haciendo plugins más complejos y potentes que en el caso anterior, al no estar limitado a una estructura predefinida. De este modo, el usuario puede utilizar bases de datos, archivos, comunicación a través de la red o cualquier otro recurso que esté disponible para el lenguaje Java.

Aun así el usuario tiene unas sencillas restricciones:

- La clase Java del plugin debe extender de la clase *jopasPlugin*.
- El constructor de la clase debe recibir como parámetros de entrada una referencia, una instancia de la clase *Jopas* y una referencia de la clase de tipo *JFrame*.

- Debe implementar los métodos *getCategory* y *getTitle*.

La clase Java debe extender de la clase *jopasPlugin* entregada entre las librerías de la aplicación. Esta clase implementa una serie de métodos necesarios para que cualquier clase Java que extienda la misma se convierta en un plugin compatible con easySP.

Los joPAS-Plugin deben disponer de los métodos *getCategory* y *getTitle* para que easySP lea los parámetros categoría y título de la aplicación del plugin al inicio. Tras invocar dichos métodos easySP es capaz de listar todos los plugins instalados en la aplicación y ordenarlos por categorías.

Además el constructor de la clase debe recibir como parámetros una referencia al API joPAS en ejecución y una referencia al *JFrame* en el que se visualizará el plugin, ya que el plugin implementado por el usuario es un panel que se visualiza dentro de easySP. easySP utiliza la API joPAS para realizar las llamadas a Octave y ejecutar las funciones de procesado de señal implementadas por el usuario, es recomendable que tan solo se cree una instancia de la clase joPAS. Esto es necesario para optimizar el uso de recursos por parte de la aplicación, así que se obliga a todos los plugins, tanto los realizados mediante de ficheros XML como los joPAS-Plugins, a utilizar la misma instancia de joPAS. El usuario de los XML-Plugins no es consciente de la utilización de joPAS, ya que él no codifica ninguna línea de Java, pero los usuarios de los joPAS-Plugins deben utilizar dicha API.

Como ya se ha mencionado anteriormente joPAS fue creado para servir de nexo de unión entre Java y Octave, y es una API de Java optimizada para la obtención de un rendimiento óptimo de las aplicaciones que hacen uso de la misma y para facilitar el uso de la API al programador.

Si bien es cierto que gracias a joPAS se pueden crear directamente aplicaciones Java de procesado de señal realizando los cálculos matemáticos en Octave. Los autores de joPAS recomendamos su uso tanto para la realización de proyectos de fin de carrera como para el desarrollo y prototipado rápido de proyectos de investigación. Pero para la enseñanza de procesado de señal es mucho más recomendable el uso de easySP, empleando cualquiera de los dos tipos de plugins, dependiendo la complejidad y necesidades del plugin a implementar.

Aunque un joPAS-Plugin es un panel de la aplicación, podrá disponer de tantos diálogos y ventanas como considere oportuno el desarrollador, ya que dichos aspectos no están limitados por easySP. Pero todo plugin parte de un panel de la aplicación principal para dar una sensación de uniformidad y facilitar la comprensión del usuario.

V. RESULTADOS

Como resultado de la ejecución de los objetivos especificados se ha conseguido la herramienta easySP, cubriendo el aprendizaje del procesado de señal desde los dos niveles de profundidad planteados. easySP a nivel usuario

resulta una aplicación realmente sencilla de utilizar. Una vez que el estudiante selecciona el módulo con el que quiere practicar, visualiza una ventana como la que se puede ver en la figura 2.

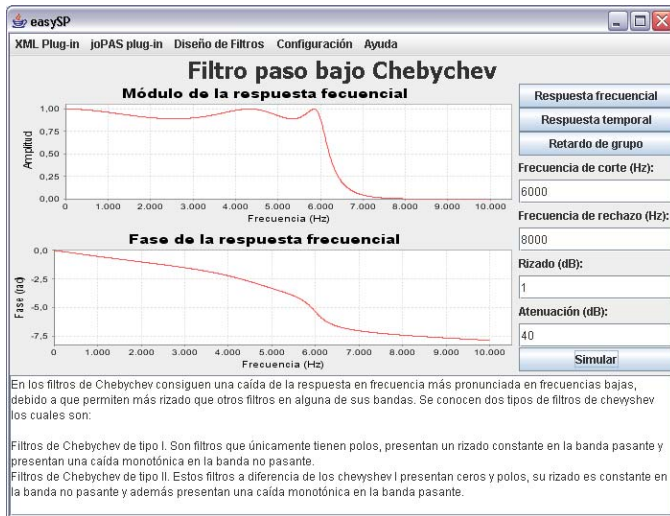


Figura 5. Simulación de un filtro Chebychev con frecuencia de corte de 6kHz

El ejemplo de la figura 2, corresponde con una simulación de un filtro paso bajo Chebychev, en el que el usuario puede modificar los parámetros fundamentales que definen las características del filtro y analizar el comportamiento del mismo en función de la variación de dichos parámetros. En la parte inferior de la ventana, el alumno encuentra una explicación teórica del comportamiento del filtro, que puede ser verificado realizando diferentes pruebas.

En este caso es interesante para el alumno, comprobar que el filtro Chebychev tiene un rizado constante en la banda de paso, y cómo la banda de transición se hace más restrictiva el orden del filtro aumenta, elevando como consecuencia el número de oscilaciones del rizado de la banda de paso (Figura 5). Una vez que el estudiante ha comprendido los conceptos teóricos que se le están planteando en el módulo, puede pasar a practicar con otro, asimilando de una forma muy amena los conceptos del procesado de señal.

En el segundo nivel de profundidad, los alumnos pueden desarrollar nuevos módulos siguiendo la estructura XML definida en la figura 6, o analizar los ya existentes, para comprobar las sentencias de Octave que son necesarias para implementar el módulo planteado. En la figura 6, se puede observar la estructura del fichero XML necesaria para realizar la parametrización del ejemplo del filtro paso bajo de la figura 2. Como se puede comprobar la creación de este fichero, que define la estructura del interfaz de usuario, es sencilla y fácil de comprender.

El estudiante puede analizar todo el proceso del cálculo de la simulación del filtro que se implementa en la función de Octave "filterDesign" (Figura 7), que contiene todas las instrucciones necesarias para el cálculo de la respuesta frecuencial, la respuesta al impulso, la respuesta al escalón y el retardo de grupo del filtro. Esta función tiene como

parámetros de entrada las variables definidas en los elementos *Input* y devuelve las variables que manejan los elementos *Graph*.

```

funcion [m,p,F,s,i,g,t]=filterDesign(fs,fp,R,A)
Fs=20000;
[n,w]=cheb1ord(fs/(Fs/2),f/(Fs/2),R,A);
[b,a]=cheby1(n,R,w);
[H,F]=freqz(b,a,1024,Fs);
m=abs(H);
p=unwrap(angle(H));
[i,t]=impz(b,a,[],Fs);
s=filter(b,a,ones(1,length(t)));
g = grpdelay(b,a,1024);
Endfuncion;

```

Figura 6. Función de Octave para el diseño del filtro Chebyshev.

Si el alumno quisiera implementar un nuevo módulo que realice lo mismo pero para un filtro Butterworth, el fichero XML a utilizar es prácticamente el mismo, solo tiene que cambiar el Campo Title, la descripción teórica del módulo, la función de Octave a la que se invoca y escribir dicha rutina (figura 7). El resultado de las implementaciones de los distintos algoritmos en XML se puede observar en las figuras 8 y 9.

```

funcion [m,p,F,s,i,g,t]=filterDesign2(fc,fs,R,A)
Fs=20000;
[n,w]=butter(fc/(Fs/2),fs/(Fs/2),R,A);
[b,a]=butter(n,w);
[H,F]=freqz(b,a,1024,Fs);
m=abs(H);
p=unwrap(angle(H));
[i,t]=impz(b,a,[],Fs);
s=filter(b,a,ones(1,length(t)));
g = grpdelay(b,a,1024);
Endfuncion;

```

Figura 7. Función de Octave para el diseño del filtro Butterworth.

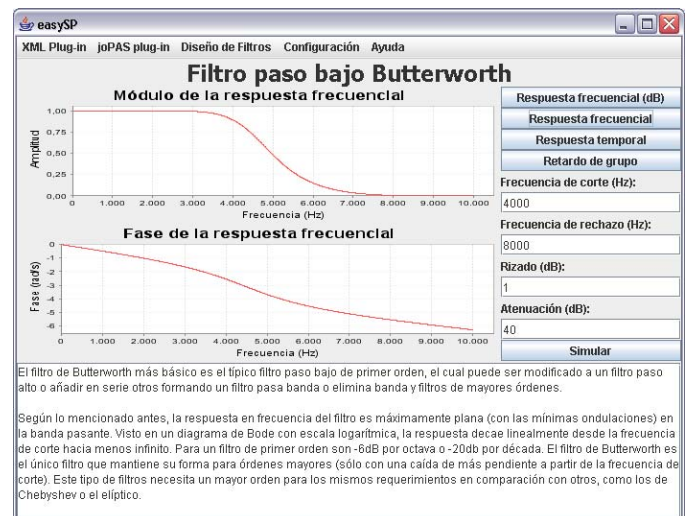


Figura 8. Simulación de un filtro Butterworth con frecuencia de corte de 4KHz

Además se proporcionan procesados más avanzados en forma de joPAS-Plugins, para que el usuario que deba implementar este tipo de plugins pueda aprender a crearlos de una forma más sencilla.

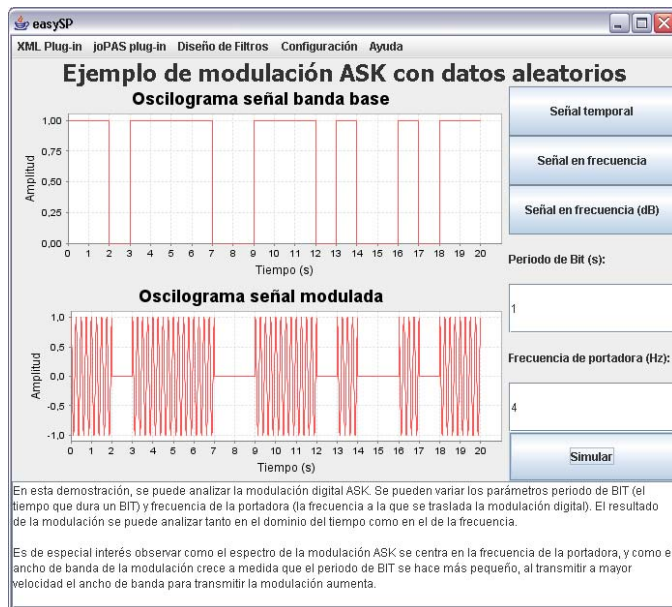


Figura 9. Ejemplo de modulación ASK

Junto con los resultados funcionales, es necesario evaluar la eficiencia sobre el aprendizaje de nuestros estudiantes de la herramienta desarrollada. Esto ha sido posible gracias a una encuesta de opinión anónima (cuyos ítems se pueden ver en la tabla II), realizada sobre un grupo de control de 10 becarios del laboratorio de telecomunicaciones de la Universidad de Deusto.

Las 10 personas con las que se ha realizado el estudio, habían cursado con anterioridad la asignatura de Tratamiento Digital de la Señal de la forma tradicional, por lo tanto, tenían todos los conocimientos necesarios, y además, podían opinar sobre la utilidad de la herramienta.

Los ítems han sido evaluados del 1 al 5, siendo 5 la puntuación más alta. Los resultados obtenidos de la encuesta se pueden ver en la tabla II, y su interpretación se ha plasmado en las siguientes afirmaciones:

1. Aunque el uso de la aplicación es intuitivo, al principio no resulta muy usable, debido a que todavía se encuentra en fase de mejora, en lo que se refiere al interfaz gráfico y facilidad de uso.
2. Se confirma que las demostraciones proporcionadas en la aplicación permiten asimilar de una forma más sencilla la teoría de las asignaturas.
3. Los XML-plugins resultan más fáciles de implementar que los joPAS-plugins, ya que para los segundos se necesita tener amplios conocimientos de programación JAVA.
4. Los XML-plugins son más sencillos, pero los alumnos perciben que se pueden implementar

plugins más completos y versátiles utilizando joPAS-plugins.

El hecho de tener que diseñar una demostración permite profundizar en la materia. El alumno se encuentra más motivado al seguir la metodología planteada con esta herramienta.

TABLA II. Resultados de la encuesta de satisfacción

ITEM	NOTA
¿Es la aplicación intuitiva?	4.1
¿Es fácilmente usable?	3.5
¿Los plugins de la aplicación facilitan la comprensión de los contenidos teóricos?	4.5
¿Es alto el tiempo empleado en dominar la aplicación?	2
Complejidad del diseño de XML-plugin	3
Funcionalidad que se puede implementar con XML-plugins	3.5
Complejidad del diseño de joPAS-plugin	4.5
Funcionalidad que se puede implementar con joPAS-plugins	4
¿El diseño de plugins permite profundizar en los contenidos de la asignatura?	4.5
¿Te resulta más motivador diseñar plugins que realizar prácticas convencionales?	4.5
Satisfacción general sobre la aplicación	4

VI. CONCLUSIONES

La aplicación desarrollada ha facilitado enormemente tanto la labor del profesor como la del estudiante, permitiendo no sólo asimilar contenidos sino desarrollar la creatividad. El profesor dispone de una herramienta versátil que le permite transmitir los conocimientos de procesado digital de señal, mediante la implementación de pequeños plugins como soporte a los contenidos teóricos que desee impartir [16][17].

Los alumnos asimilan mejor los conocimientos de procesado mediante una herramienta con la que pueden interactuar [18][19][20]. Además pueden desarrollar nuevos módulos sin tener conocimientos de otros lenguajes de programación con la excepción de Octave.

Actualmente, se está trabajando en un asistente que genere los contenidos del XML para que el alumnado no necesite especificar el fichero de configuración y pueda dedicar todo su esfuerzo en la implementación del algoritmo en Octave.

Además, se permite la creación de joPAS-Plugins, haciendo que el usuario con conocimientos de Java pueda crear plugins con nuevos sistemas (filtros, moduladores, etc.) de una potencia mucho mayor.

Como conclusión final, se puede destacar que el resultado de esta API ha sido muy satisfactorio, para desarrolladores y usuarios, ya que proporciona una herramienta muy potente para el rápido desarrollo de aplicaciones de procesado de señal, siendo su único inconveniente, que se debe conocer el lenguaje Java para realizar los plugins más avanzados.

AGRADECIMIENTOS

Los autores de este artículo queremos agradecer a los alumnos por transmitirnos sus experiencias con la aplicación y por contribuir activamente en la ampliación de la misma mediante la aportación de nuevos plugins desarrollados por ellos.

REFERENCIAS

- [1] Begoña García, Javier Vicente, "Herramienta para la Simulación e Implementación Real de Sistemas Discretos FIR e IIR" in *Proc. TAEE'02*, Las Palmas, Spain, 2002.
- [2] J. Angulo, B. García, J. Vicente, I. Angulo, "Microcontroladores avanzados dsPIC", International Thomson Editores, 2005.
- [3] B.L. Sturm, J.D. Gibson, "Signals and Systems using MATLAB: an integrated suite of applications for exploring and teaching media signal processing", in *Proceedings 35th Annual Conference Frontiers in Education, FIE '05*.
- [4] M Murphy, "Octave: A Free, High-Level Language for Mathematics" in *Linux Journal*, 1997.
- [5] Y Cheneval, L Balmelli, P Prandoni, J Kovacevic, M Vetterli, "Interactive DSP education using Java" in *ICASSP'98*, Seattle, WA, USA, 1998.
- [6] WS Gan, YK Chong, W Gong, WT Tan, "Rapid prototyping system for teaching real-time digital signalprocessing" in *IEEE Transactions on Education*, 2000.
- [7] Kurt Hornik, Friedrich Leisch, Achim Zeileis, "Ten Years of Octave Recent Developments and Plans for the Future" in *Proc. DSC 2003*, Vienna, Austria, 2004.
- [8] JW Eaton, "Octave: Past, Present, and Future" in *DSC 2001*, Vienna, Austria, 2001.
- [9] NC Marques, F Azevedo, C Morgado, JF Custódio, "Using Octave to introduce programming to technical science students" in *SIGCSE 2005*, Caparica, Portugal, 2005.
- [10] Javier Vicente, Begoña García, Amaia Mendez, Ibon Ruiz, Oscar Lage, "Teaching Signal Processing Applications With joPAS: Java To Octave Bridge" in *Proc. EUSIPCO 2006*, Firenze, Italy, 2006.
- [11] Muthiah Annamalai, Hemant Kumar, Leela Velusamy, "Octave-GTK: A GTK binding for GNU Octave", *The Cornell University Library*, 2006.
- [12] Ken Arnold and James Gosling, "The Java Programming Language" *The Java Series*. Addison-Wesley, Reading, MA, 1996.
- [13] E. Roberts, "Resources to support the use of Java in introductory computer science" in *35th SIGCSE technical symposium on Computer science education*, Norfolk, Virginia, USA, 2004.
- [14] P Faggion, J An, E Ambikairajah, "Online Java signal processing education" in *Proc. ICICS 2001*.
- [15] Clemens H. Cap, "XML goes to School: Markup for Computer Assisted Learning and Teaching" in *The European Journal of Open and Distance Learning*, 2000.
- [16] J Campbell, F Murtagh, M Köküer, "DataLab-J: A Signal and Image Processing Laboratory for Teaching and Research", in *IEEE Transactions on Education*, 2001.
- [17] WS Gan, "Teaching and Learning the Hows and Whys of Real-Time Digital Signal Processing" in *IEEE Transactions on Education*, 2002.
- [18] PSR Diniz, "Adaptive Filtering: Algorithms and Practical Implementation", *Kluwer Academic Publishers*, 2002.
- [19] A Spanias, S Urban, A Constantinou, M Tampi, et al., "Development and evaluation of a Web-based signal and speechprocessing laboratory for distance learning", in *Proceeding of IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP '00*.
- [20] D Sage, M Unser, "Teaching image-processing programming in Java", *IEEE Signal Processing Magazine*, 2003.



Javier Vicente Sáez nació en Baracaldo (Vizcaya) en 1978. Se graduó en Ingeniería Técnica Industrial, especialidad electrónica en 1999 por la Universidad de Deusto. Posteriormente, en 2001 se graduó en Ingeniería de Telecomunicaciones, por la Universidad de Deusto. Consiguió la suficiencia investigadora después de realizar los correspondientes cursos de doctorado en la Universidad de Deusto (UD), Bilbao, en 2005, y tiene registrada la tesis doctoral en nuevas

arquitecturas de procesado digital de señal.

Desde el año 2001 es profesor y forma parte del Departamento de Telecomunicaciones de la UD. En el año 2001, crea junto a Dra. Begoña García Zapirain el grupo investigación Procesado Avanzado de Señal (PAS) de la UD.



Begoña García Zapirain nació en Donosti (Gipuzkoa) en 1970. Se graduó en Ingeniería de Telecomunicación especialidad Telemática, por la Universidad del País Vasco en 1994. En 2003 presento su tesis Doctoral en el campo del procesado digital de voces patológicas.

Tras unos años de trabajo en la empresa ZIV, en 1997 se incorpora al claustro de la Universidad de Deusto como profesora en el ámbito de la teoría de señal y electrónica. Desde el año 2002 dirige el Departamento de Telecomunicaciones de la UD. En el año 2001, crea junto a Javier Vicente Sáez el grupo investigación Procesado Avanzado de Señal (PAS) de la UD, en el que desarrolla el papel de investigadora principal. Miembro del IEEE y del EURASIP, participa en el comité organizador de TAIMA.



Ibon Ruiz Oleagordia nació en Bilbao (Vizcaya) en 1975. Se licenció en Ciencias Físicas en la Universidad del País Vasco (UPV), Bilbao, en 1999 y obtuvo el título de Ingeniero en Electrónica en la misma universidad en el año 2001. Consiguió la suficiencia investigadora después de realizar los correspondientes cursos en la Universidad de Deusto (UD), Bilbao, en 2002, y tiene registrada la tesis doctoral.

Desde el año 2000 es profesor y forma parte del Departamento de Arquitectura de los Computadores, Electrónica, Automática y Telecomunicaciones de la UD. En el año 2002, entra a formar parte del grupo investigación Procesado Avanzado de Señal (PAS) de la UD.



Amaia Méndez Zorrilla nació en Baracaldo (Vizcaya) en 1978. Se graduó en Ingeniería Técnica Industrial, especialidad electrónica en 1999 por la Universidad de Deusto. Posteriormente, en 2001 se graduó en Ingeniería de Telecomunicaciones, por la Universidad de Deusto. Consiguió la suficiencia investigadora después de realizar los correspondientes cursos de doctorado en la Universidad de Deusto (UD), Bilbao, en 2005, y tiene registrada la tesis doctoral en

ingeniería biomédica.

Desde el año 2003 es profesora y forma parte del Departamento de Telecomunicaciones de la UD. En el año 2005 entra a formar parte del grupo investigación Procesado Avanzado de Señal (PAS) de la UD.



Oscar Lage Serrano nació en Baracaldo (Vizcaya) en 1982. Se graduó en Ingeniería Técnica en Informática de Gestión en 2003 por la Universidad de Deusto. Posteriormente, en 2005 se graduó en Ingeniería Informática, por la Universidad de Deusto. En el año 2005, entra a formar parte del grupo investigación Procesado Avanzado de Señal (PAS) de la UD como doctorando de investigación.