CONSTRUCCIÓN DE UNA HERRAMIENTA DIDÁCTICA EN JAVA PARA LA GRAFICACIÓN DE CÓDIGOS DE LÍNEA

Jaime Eduardo Trujillo Ruiz
Especialista en Redes y Comunicaciones
Especialista en Finanzas
Ingeniero Electrónico
Docente Universidad Católica Popular del Risaralda
jaimetrujillo@ucpr.edu.co

Héctor Armando Arredondo López
Estudiante VI Semestre de Ingeniería de Sistemas y Telecomunicaciones
manditopez@yahoo.com

Juliana Bermúdez Henao
Estudiante VI Semestre de Ingeniería de Sistemas y Telecomunicaciones
julimarmudez@hotmail.com

Recibido Mayo 25 de 2008 – Aceptado Julio 08 de 2008

RESUMEN

En este articulo: “Construcción de una Herramienta Didáctica en Java para la Graficación de Códigos de Línea”, desarrollado en el lenguaje de programación Java, básicamente se describe cómo se implementa, en un programa, una herramienta para graficar códigos utilizados en las telecomunicaciones, y cómo los códigos de línea mejoran en mayor o menor medida la calidad de la señal recibida. En cada código se presentan su técnica y algunas gráficas.

Palabras Clave: Códigos de línea, java, telecomunicaciones.

ABSTRACT:

This article: “Construction of a Teaching Tool in JAVA for Line Codes Graph” developed in Java programming language, basically describes the way to implement, in software, a tool to graph codes used in telecommunications, and how the line codes improve to greater or lesser extent the quality of the signal received. Its technique and some graphics are presented in each code.

Key Words: Line Codes, java, telecommunications.

1. INTRODUCCIÓN

Existen diferentes factores que se deben evaluar para la adecuada recepción de señales digitales, tales como la velocidad de transmisión, el ancho de banda, la relación energía de bit/ ruido promediado (Eb/No), entre otros, ya que cada uno de ellos afecta en diferente medida la transmisión, algunas conclusiones que se pueden sacar de la evaluación de estos factores son:

- Un aumento en la velocidad de transmisión aumenta la tasa de error de bit (BER, medida habitual para la cantidad de errores que se producen en una línea de transmisión, su lectura se hace en términos probabilísticos. Si se tiene una BER de 10^{-6} se interpreta como la posibilidad de tener un error por cada millón de bits transmitidos).
- Un aumento en la relación señal ruido (Eb/No) reduce la tasa de error por bit.
- Si se aumenta el ancho de banda se aumenta la velocidad de transmisión.

Otro factor importante y el que le da la naturaleza a esta discusión es el denominado: códigos de línea.

El presente artículo se encarga de explicar las diferentes técnicas para codificar líneas y cómo a través de una herramienta de desarrollo como Java se puede estructurar una

aplicación académica para el entendimiento de los métodos de transmisión de bits o códigos de línea.

2. CÓDIGOS DE LÍNEA

Los códigos de línea son técnicas de transmisión de datos digitales en forma de señales digitales para realizar un método de comunicación netamente digital. Son característicos del nivel físico del modelo OSI.

A través de ellos se pueden implementar técnicas que permiten mejorar las exigencias de recepción, las cuales se van degradando dependiendo de la calidad del tipo de medio de transmisión que transporte las señales.

En su forma más sencilla se tiene una correspondencia 1 a 1 entre bits y hertz. Codificar es representar una señal binaria con niveles de tensión o voltajes previamente determinados. El caso trivial es asignar un voltaje positivo al uno (1) y una ausencia de voltaje al cero (0).

2.1 Criterios para comparar las técnicas de codificación o códigos de línea.

a) Espectro de la señal:

- Reducir las componentes de alta frecuencia redunda en menores necesidades de ancho de banda.
- La ausencia de componentes DC facilita la sincronización y evita la necesidad de líneas directas.
- La potencia transmitida se debe concentrar en la parte central del ancho de banda y no en las partes laterales ya que esto incrementa la distorsión.

b) Sincronización:

Se refiere a conocer el principio y final de cada bit.

c) Detección de errores:

Es una ventaja si se tiene un esquema para detectar errores desde el mismo código usado, ayudando a las capas superiores del modelo a la detección de los mismos.

d) Inmunidad al ruido e interferencia:

La posibilidad de evaluar y diferenciar los códigos de línea de acuerdo con su inmunidad al ruido, el criterio de comparación usado es la BER.

e) Costo y complejidad:

A mayor velocidad de transmisión mayor costo de la electrónica digital.

3. TIPOS DE CÓDIGOS DE LÍNEA.

En el presente artículo se discutirán cuatro metodologías para realizar códigos de línea.

3.1) Unipolares: La señal es representada con ausencia de tensión para el cero (0) y un nivel positivo o negativo para el uno (1).

Presenta grandes desventajas como la no anulación del nivel DC, no tiene capacidad de sincronización y requiere de líneas directas.

3.2) Polares: Las señales binarias siempre son representadas por un nivel de tensión, no se usa el cero (0) o ausencia de tensión.

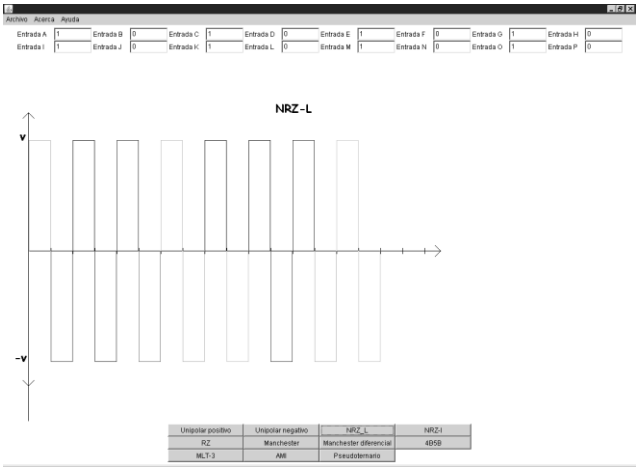
A su vez los códigos polares se dividen en tres tipos:

3.2.1) NRZ: Códigos sin retorno a cero

- **NRZ-L**: Sin retorno a nivel cero.

Técnica: Se usa un nivel negativo de voltaje para representar un estado binario y un nivel positivo para representar al otro. Este código presenta ventajas como uso eficiente del ancho de banda y fácil implementación. Su principal desventaja es la presencia de componente DC continua, aunque el alternar los niveles de tensión es una buena opción. Hay incertidumbre

en el número de unos y ceros, además no tiene capacidad de sincronización. Un ejemplo de NRZ –L se presenta en la gráfica 1.



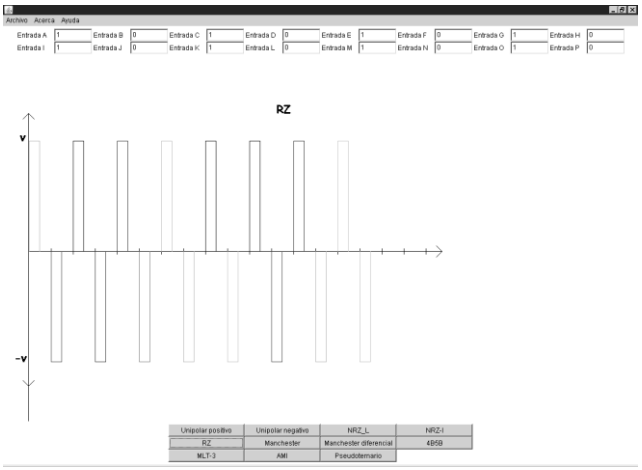
Gráfica 1

- **NRZ-I**: No retorno a cero con inversión de unos.

Técnica: Los datos se codifican mediante la presencia o ausencia de una transición de la señal al principio del intervalo del bit. Para un cero (0) se codifica con la misma señal que el bit anterior. Para un uno (1) se realiza una transición con una señal diferente al bit anterior. Entre sus principales ventajas está el hecho de presentar mejor eficiencia ante la presencia de ruido ya que es más fácil detectar una transición de señal que un valor de tensión comparado con un umbral. Su principal desventaja es la sincronización, una pérdida del reloj en transmisor y receptor puede repercutir en la imposibilidad de detectar el inicio de un bit ante una secuencia de ceros larga

3.2.2) RZ: Códigos con retorno a cero (0). Se caracterizan porque el nivel de tensión no es constante durante la duración del bit.

Presentan como ventaja la facilidad de sincronización gracias a las transiciones, en detrimento del ancho de banda. Un ejemplo de codificación RZ se presenta en la gráfica 2.



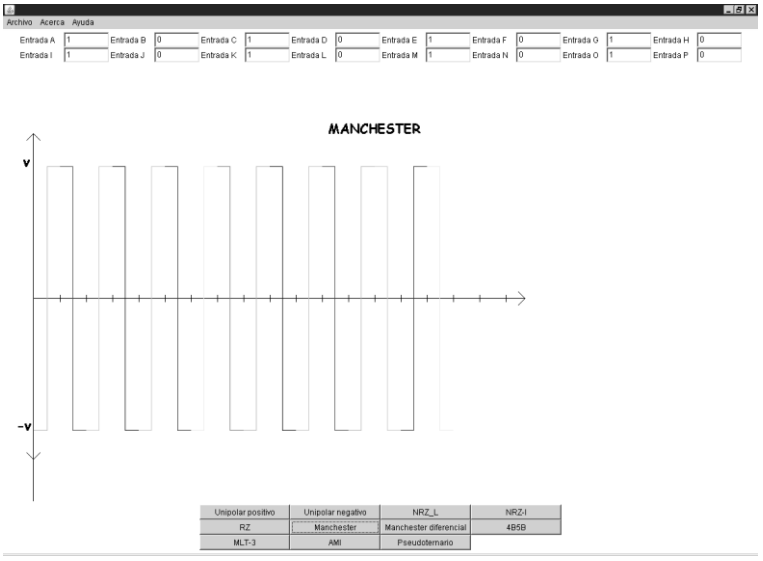
Gráfica 2.

3.2.3) Bifásicos: Se definen como aquellos códigos en los que siempre hay una transición en mitad del intervalo de duración del bit.

- **Manchester:**

Técnica: Una transición de un nivel bajo de voltaje (negativo) a un nivel alto (positivo) representa un uno (1) binario.

Una transición de un nivel alto de voltaje (positivo) a un nivel bajo (negativo) representa un cero (0) binario. Un ejemplo de codificación Manchester se muestra en la gráfica 3.



Gráfica 3

- **Manchester Diferencial:** En este caso la técnica usada es: Un cero (0) se representa por una transición al principio del bit y un uno (1) se representa por ausencia de transición al principio del intervalo de duración del bit.

3.2.4) Un tipo especial de código polar es el 4B/5B, corresponde al tipo de código mBnB, donde m Bits de datos generan n bits de código, en este caso la eficiencia de la transmisión es del 80% ya que 4 bits de datos de usuario generan un código de 5 bits.

Técnica:

Inicialmente se agrupan bloques de 4 bits, posteriormente se asigna una representación de 5 bits al bloque de 4 bits de acuerdo con la tabla 1 que está previamente establecida y finalmente el código de 5 bits se codifica con patrones NRZ-I.

CODIFICACIÓN 4B/5B			
Datos de Entrada (4 bits)	Grupo de Código (5 bits)	Datos de Entrada (4 bits)	Grupo de Código (5 bits)
0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

Tabla 1

Los códigos bifásicos presentan ventajas como la sincronización ya que el receptor puede usar las transiciones para sincronizarse adecuadamente, elimina las componentes continuas, las ausencias de transición son un primer nivel para la detección de errores ante la no existencia de éstas en mitad del intervalo.

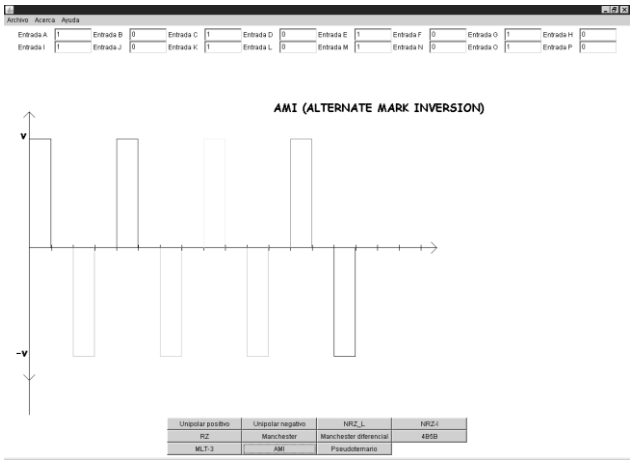
Su principal desventaja es que requieren mayor ancho de banda.

3.3) Bipolares: Son aquellos en los que las señales binarias pueden tener tres estados, niveles positivos o negativos de voltaje y cero (0) o ausencia de señal.

Como ejemplo de códigos bipolares se pueden relacionar tres:

3.3.1) AMI: Usa una técnica en la cual el cero (0) se representa por ausencia de señal y un uno (1) se representa por un pulso negativo o positivo de forma alternada.

AMI cuenta con ventajas como evitar problemas de sincronización ante secuencias largas de unos (1) gracias a las transiciones, elimina el nivel DC y genera un ancho de banda menor comparado con otros códigos. Como desventaja tiene que las secuencias largas de ceros (0) siguen siendo un problema. Un ejemplo de AMI se muestra en la gráfica 4.



Gráfica 4.

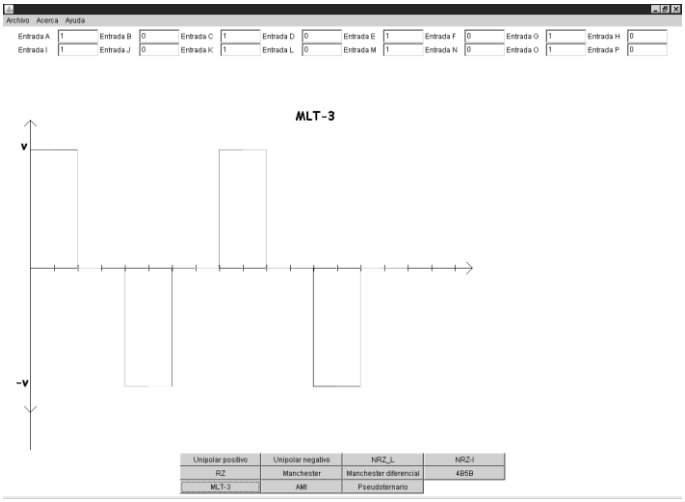
3.3.2) MLT-3: Es un código que usa transiciones que pueden tener tres estados: voltaje positivo, voltaje negativo y ausencia de voltaje.

Técnica:

- Si el siguiente bit de entrada es cero (0), el siguiente valor de salida es el mismo que el del valor precedente.
- Si el siguiente bit de entrada es uno (1), el siguiente valor de entrada implica una transición:

Si el valor de salida anterior fue positivo o negativo el siguiente valor de salida es cero (0).

Si el valor de salida precedente fue cero (0), el siguiente valor es distinto de cero y de signo opuesto al de la última salida distinta de cero (0). En la gráfica 5 se muestra un ejemplo sencillo de codificación MLT-3.



Gráfica 5

3.3.3) SEUDOTERNARIO: Usa la misma técnica que AMI solo que los patrones se invierten.

3.4) Técnicas de Scrambling

Su objetivo es reemplazar las secuencias de bits que den lugar a niveles de tensión constante por otras secuencias que proporcionen suficiente número de transiciones de forma tal que el reloj del receptor pueda mantenerse sincronizado. Entre sus ventajas está que elimina completamente los niveles DC ante las transiciones, evita secuencias largas que corresponden a señales de tensión nula, no reducen la velocidad de transmisión de datos y

finalmente y muy importante es que desde el mismo código se pueden manejar errores en la transmisión.

Tipos de Scrambling:

3.4.1) HDB3: High Density Bipolar 3-zeros.

Se realiza sustitución de acuerdo con la tabla 2 ante secuencias de cuatro ceros consecutivos**.** Cuando no usa “violaciones de código” se codifica con AMI.

POLARIDAD DEL PULSO ANTERIOR	**NÚMERO DE PULSOS BIPOLARES (UNOS) DESDE LA ÚLTIMA SUSTITUCIÓN**	
	IMPAR	**PAR**
-	000-	+00+
+	000+	-00-

Tabla 2

3.4.2) B8ZS: (Bipolar with 8-zeros Substitution).

Está basado en AMI Bipolar, y se usa para corregir secuencias largas de ceros (0).

Técnica: Ante la aparición de 8 ceros (0) consecutivos, dicha secuencia se debe reemplazar por alguna de las siguientes violaciones según el caso:

- 000+-0-+, si el valor anterior al octeto fue negativo.
- 000-+0+-, si el valor anterior al octeto fue positivo.

4. Procedimiento de desarrollo del software en JAVA

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems, es una herramienta de programación como C, C++, BASIC, Pascal o Logo, que sirve para crear aplicaciones informáticas. Algunas de sus características fundamentales son:

- Una misma aplicación puede funcionar en diversos tipos de computadores o maquinas y sistemas operativos (Windows, Linux)... así como en otros dispositivos inteligentes.
- Los programas en Java pueden ser aplicaciones independientes, es decir que corren en una ventana propia como los "applets".
- Es un lenguaje "orientado a objetos", esto significa que los programas se construyen a partir de módulos independientes que se pueden transformar o ampliar fácilmente.

Así entonces, se pueden describir las herramientas de java en las cuales fue desarrollado el codificador. En primer lugar se utiliza el Java Compiler JDK-6 para Windows XP, esta es la plataforma que permite ejecutar todas las aplicaciones de Java. Así mismo este es el primer programa o ejecutable que se debe tener instalado en la máquina antes de comenzar a programar en Java. La aplicación o el compilador usado es el JCreator de Xinox software, este es muy sencillo y fácil de utilizar para las personas que apenas están comenzando en este campo de la programación en lenguaje Java, debido a que la aplicación está desarrollada en un código bastante sencillo para cualquier programador y consiste básicamente en capturar las entradas binarias para su posterior comparación y graficación.

Comencemos por definir las librerías básicas de Java que se utilizaron, estas son la javax.swing.*; la java.awt.*; y la java.awt.event.*. El paquete Swing es parte de la JFC (Java Foundation Classes) en la plataforma Java. Abarca componentes independientes del sistema operativo como botones, tablas, marcos, entre otras herramientas. Todo esto redunda en una mayor funcionalidad en manos del programador, y en la posibilidad de mejorar en gran medida las interfaces gráficas de usuario. Las componentes de Swing

utilizan la infraestructura de AWT, incluyendo el modelo de eventos AWT, el cual rige como una componente que reacciona a eventos de teclado, mouse, etc. Es por esto, que la mayoría de los programas Swing necesitan importar dos paquetes AWT: *java.awt.** y *java.awt.event.**. Por su parte entonces AWT es una biblioteca de clases Java para el desarrollo de Interfaces de Usuario Gráficas. Su estructura básica esta compuesta por componentes como los Buttons, Labels, los contenedores que almacenan componentes, los gestores de posición, que ubican los componentes dentro de los contenedores y los eventos, que indican las acciones del usuario.

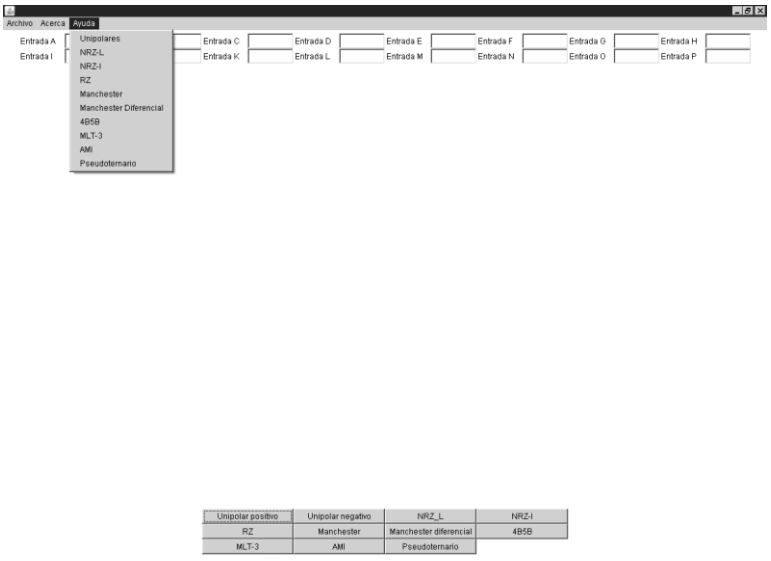
En segundo lugar se destacará que la aplicación se ejecuta como una ventana. Esta es una aplicación que se pone en marcha de forma local, y que utiliza como interfaz de usuario las tecnologías de las ventanas típicas de los sistemas operativos. Para crear la ventana de nuestro programa se utilizó el Extends Frame que ofrece AWT, en el cual se puede modificar el titulo o “super”, la posición del ángulo derecho superior de la ventana en el escritorio y especificar la anchura y altura de la misma. Posteriormente se describe la clase **Panel** que es el más simple de los contenedores de componentes gráficos. Es el componente de agrupamiento más común, su trabajo consiste en ser el contenedor del resto de los componentes del programa. Como en este caso, es tarea común poner en un frame una serie de paneles, cada uno de estos enfocado a una tarea específica. Su layout predeterminado es FlowLayout, aquí utilizamos el GridLayout ya que este proporciona gran flexibilidad para situar componentes. Este se crea con un número de filas y columnas y los componentes van dentro de las celdas de la tabla así definida. Así entonces se crean los cuatro paneles del codificador, estos son:

- p1.setLayout(new GridLayout(3,2)); En este panel se posicionan los botones para la selección del algoritmo de codificación.

- p2.setLayout(new GridLayout(2,3)); en este panel van los diferentes campos de texto y sus etiquetas.
- En el panel 3, p3 simplemente agregamos p1.
- En el panel 4, p4, agregamos p2.

De esta forma es más fácil posicionar luego los paneles en el contenedor, uno en el norte o en la parte superior de la ventana y otro en el sur o parte inferior. Aparte de los paneles también se deben tener en cuenta las variables Button, TextField y Label. Estas permiten crear un botón con un nombre específico, crear un campo de texto y un nombre para un campo de texto o etiqueta. Para el codificador se usaron, relativamente, un extenso número de botones, que van desde ba hasta bl, campos de texto y etiquetas desde ta hasta tp y etiquetas desde la hasta lp respectivamente. Los botones en el codificador nos permiten seleccionar el tipo de algoritmo de codificación, y los campos de texto permiten tener las entradas binarias de la señal que se necesitan para su respectiva codificación.

Por otra parte se utiliza una más de las tantas clases de Java, esta es el menubar. En general es una jerarquía sencilla de componentes, primero se ocupa una barra de menú (menubar) que contendrá las columnas o menús (menú) y esta barra de menú (menubar) es la que se pega al frame o ventana. A su vez los menús tienen agregados menú ítems, es decir las opciones desplegables del menú. En el codificador incluimos varios menús, ellos son: **M1**: tiene agregado el menú ítem, i1, que es abrir una nueva ventana y el menú ítem i3 que es salir del programa. **M2**: tiene agregado el menú ítem i2, que es la información acerca del programa. **M3**: es la ayuda del programa, tiene agregados desde el menú ítem i4 hasta el i13, que son todas las ventanas con la respectiva información de los algoritmos. En la gráfica 6 se observan los botones en el contenedor en la parte inferior, los campos de texto y las etiquetas en el contenedor ubicado en la parte superior y los menús en la barra superior de la ventana donde están incluidos los menús ítems correspondientes:



Gráfica 6

Se debe recordar que a cada botón, menú ítem, etc. se le agrega un ActionListener que es una Interface del grupo de los Listeners (escuchadores). Esta tiene un sólo método: void action performed (ActionEvent) y se usa para detectar y manejar eventos de acción (Action Event), es decir, cuando se produce una acción sobre un elemento del programa. En otras palabras, al pulsar un botón (Button), al hacer doble clic en un elemento de lista (List), al elegir un menú (MenuItem), entre otras. Cuando ActionListener detecta una acción se genera un evento de acción (ActionEvent) en el elemento (botón). Los ActionEvent invocan el método actionPerformed(ActionEvent e) que realiza las acciones programadas ante ese evento. En el codificador se determina que al realizarse alguna operación sobre un botón o un menú, en la variable de tipo entero comp se guarda un número diferente según el botón o el menú sobre el que se realizará la acción, en otras palabras, cada que se realice una operación sobre uno de los botones se guardará en comp un número que definirá qué acción es y posteriormente esta variable será la que nos indique qué acción se debe realizar. Así entonces, se definió que comp= 1 sería la acción a realizarse si presionan el botón b1 y así

están implementadas todas secuencialmente hasta los menús. En el lenguaje Java es posible dibujar simplemente volviendo a definir el método Paint. Este método se invoca automáticamente por el sistema cuando la ventana pasa a un primer plano. Cuando se haya dibujado y se quieran visualizar posibles cambios, es posible invocar el método Repaint() el parámetro g de tipo Graphics de Paint incluye los métodos gráficos que Java puede utilizar y representar. Graphics es una clase que no se puede incluir, sino que es recibida por Paint como parámetro. Por lo tanto se puede utilizar sólo en ella, esta tiene algunas primitivas de dibujo que se pueden utilizar, en este caso se usaron: g.setColor(); g.drawLine(); g.drawString(); con estas podemos graficar todas las señales con los diferentes códigos.

5. CONCLUSIONES.

- El artículo no pretende la realización de un codificador de línea, es simplemente el uso de una herramienta de programación visual como Java para el desarrollo de un software de tipo didáctico que debe servir de apoyo al concepto de códigos de línea impartido en facultades interesadas en formar competencias fundamentales en telecomunicaciones. El software se limita a la entrada de un número limitado de bits y de acuerdo a la técnica seleccionada desarrollar el algoritmo de forma gráfica para su más fácil entendimiento.
- Otra conclusión relevante y pertinente para este texto, es la importancia de los Códigos de Línea, ya que son claves para la adecuada recepción de señales en ambientes puros digitales. Sus áreas de aplicación van desde las redes de datos locales hasta las redes de transporte. Ejemplo de ello, el código Manchester diferencial y su uso en Token Ring, MLT-3 en 100 Base Tx, 4b/5b en entornos FDDI y las técnicas de Scrambling que se usan en portadoras digitales HDB3 en la jerarquía digital europea y B8ZS en la jerarquía americana.

BIBLIOGRAFÍA.

- Aguilar, Luis Joyanes. I. Z. M. (2002). *Programación en Java 2: Algoritmos, Estructuras de Datos y Programación*. Editorial Mc Graw Hill.

- Anonymous ()… (2008). http://www.programacionfacil.com/java:menubar

- Froufe (). *Tutorial de Java*. (2008). http://www.cica.es/formacion/JavaTut/

- Haykin, Simon. *Sistemas de Comunicación*. Limusa Wiley.

- Marcello, Valeri, G. N. (2006). *Java 5- Novedades del Lenguaje*. 2008. http://books.google.com.co/books?id=t1HOYNTih_4C&client=firefox-a

- Sierra, F. J. (2002). *El Lenguaje de Programación Java.* Editorial Alfaomega.

- Stallings, William. *Comunicaciones y Redes de Computadoras*. Edición 6. Prentice Hall.

- Tomasi, Wayne. *Sistemas de Comunicaciones Electrónicas*. Edición 2. Prentice Hall.