UNIVERSIDAD AUTONOMA METROPOLITANA - AZCAPOTZALCO
DIVISION DE CIENCIAS BASICAS E INGENIERIA
MAESTRIA EN CIENCIAS DE LA COMPUTACION

REPORTE DE PROYECTO TERMINAL II

"SIMULADOR DIDÁCTICO EN 3D DE CIRCUITOS ELECTRICOS Y ELECTRONICOS"

Presenta: FERNANDO RAMIREZ ROJAS

Asesor: DR. CARLOS AVILÉS CRUZ

Octubre, 2010

CONTENIDO

MVC – Código de Clases

CP – Código de Clases

I. INTRODUCCION.

En la actualidad los dispositivos electrónicos basados en procesadores (computadoras personales, teléfonos, agendas, etc.), permiten manejar la información de manera interactiva, debido a la rapidez con que dichos dispositivos procesan la información.

Tomando en cuenta este hecho, se puede tomar ventaja de estos dispositivos como herramientas de aprendizaje.

Comparando el tipo de información “estática” que presenta un libro, con la posibilidad de presentar esa misma información de manera dinámica, interactiva y agradable, como por ejemplo los mapas interactivos de “google”, se puede experimentar la ventaja de “moverse”, “acercarse”, “alejarse” e incluso “meterse” literalmente en dichos mapas, además de poder observar en tres dimensiones algunos de ellos.

Esta es la idea similar que se pretende lograr con el proyecto que se describe en este reporte. Realizar un “mapa en 3D” para los circuitos eléctricos y electrónicos, que nos permita captar de un solo golpe de vista, la información particular y en contexto, de los principales conceptos que se manejan en los circuitos eléctricos: voltaje y corriente. Sin embargo, “un extra” que contiene este proyecto, es que no solo proporciona dichos parámetros, sino que muestra su comportamiento “dinámico”, esto es, podemos observar tanto el flujo de la corriente en un circuito en corriente directa, como el “movimiento o vibración” de todo el circuito, cuando se le golpea con una señal senoidal, lo que permite observar, por ejemplo, el concepto del “defasamiento” entre señales de corriente alterna. Es aquí, en la dinámica en 3D, donde radica lo novedoso de este simulador, aplicado a los circuitos eléctricos y electrónicos.

Con el fin de apoyar la enseñaza de los circuitos eléctricos y electrónicos, mediante la utilización de materiales didácticos visuales e interactivos que integren a la geometría espacial con dichos circuitos, se presenta el siguiente simulador didáctico que permite las siguientes funciones:

1. Dibujar un circuito eléctrico ó electrónico en un editor gráfico en dos dimensiones.
2. Simular el circuito en corriente directa (DC) y en corriente alterna (AC) a una frecuencia determinada.
3. Presentar los resultados de la simulación en DC, voltajes nodales y corrientes en cada elemento, en un escenario de tres dimensiones.
4. Presentar los resultados de la simulación en AC, magnitudes y fases de voltajes nodales, en un escenario de tres dimensiones.

El diagrama a bloques del proyecto se muestra en la Figura I.1.

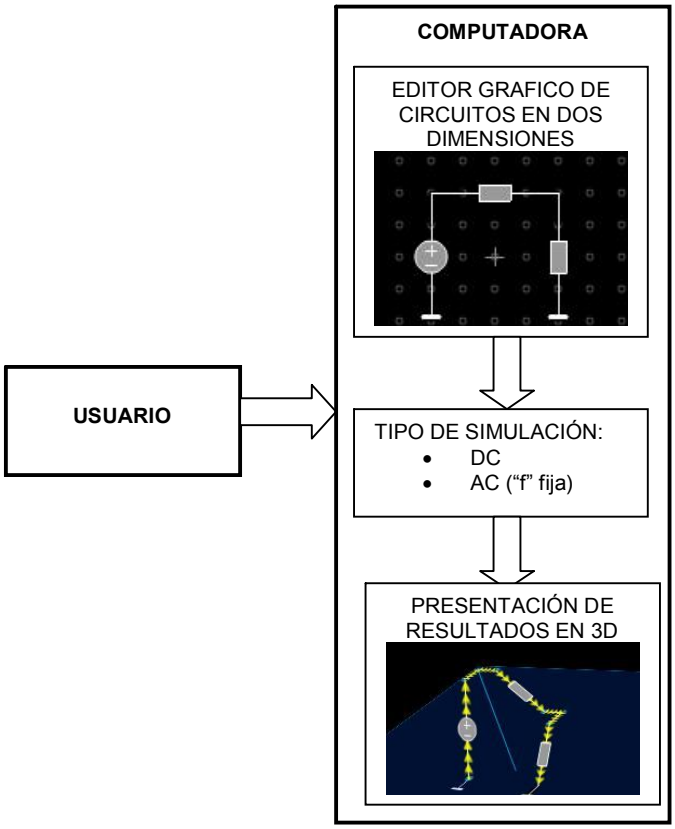


Figura I.1. Diagrama a bloques del Simulador Didáctico en 3D de Circuitos Eléctricos y Electrónicos

Los resultados de la simulación del circuito en corriente directa, se presentan en una imagen animada e interactiva en tres dimensiones (3D). El usuario podrá “girar” la imagen para observar el circuito desde diferentes ángulos, como se muestra en la Figura I.2.

En dicha figura, se presentan varias vistas en 3D de un circuito resistivo. El plano mostrado representa el plano de referencia de potencial “cero” volts o tierra. Las uniones entre los componentes del circuito, representan los nodos del mismo a diferentes potenciales (alturas) con respecto al plano de tierra, mientras que el flujo de flechas representan los flujos de corriente en cada rama de éste.

La altura o distancia de los nodos respecto al plano de referencia es proporcional al valor del voltaje nodal asociado a dicho nodo, mientras que el ancho del flujo de flechas, es proporcional al valor de la corriente de rama en cada elemento del circuito.

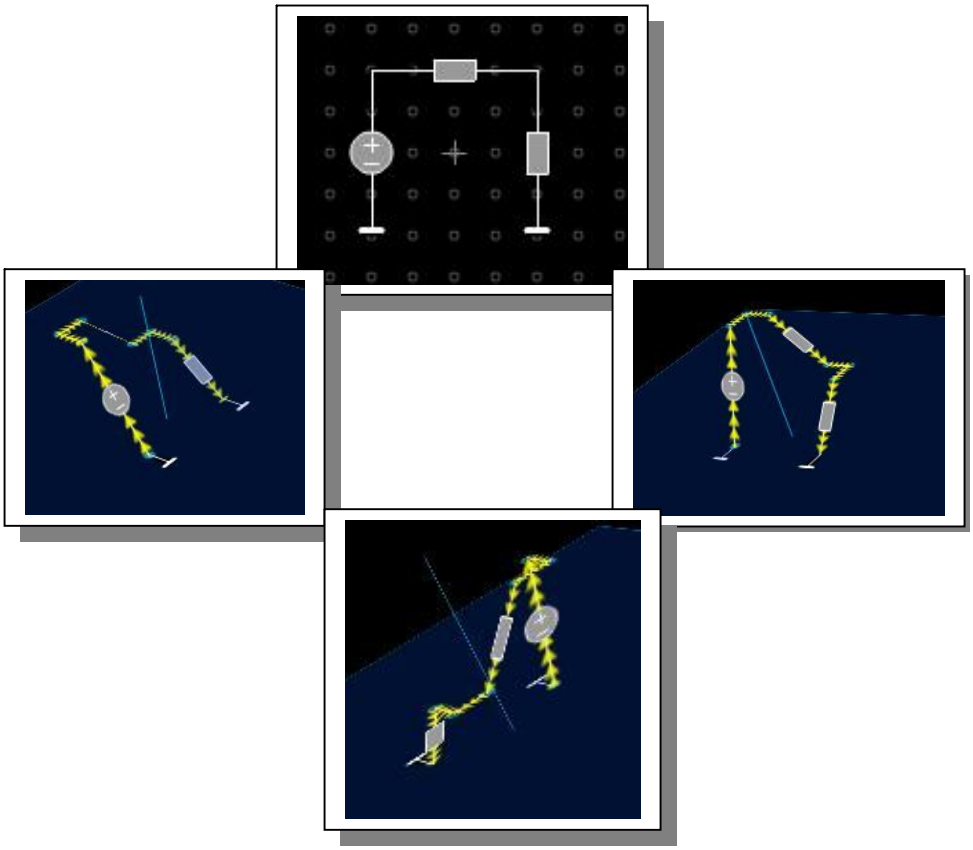


Figura I.2. Diferentes vistas de la representación en 3D de los voltajes nodales y de las corrientes en un circuito formado por dos resistencias y una fuente de voltaje de corriente directa.

Además de las graficas anteriores, en la simulación en DC se presenta una gráfica de barras, la cual representa el valor de los voltajes nodales en forma de barras interactivas, es decir, cuando se coloca el mouse sobre alguna barra, ésta se ilumina, se reporta el valor de dicho voltaje y el número de nodo asignado a éste, mientras que, al mismo tiempo, se resalta mediante un círculo concéntrico, el nodo correspondiente en la gráfica en 3D. En la Figura I.3 se muestra un ejemplo de lo explicado anteriormente.

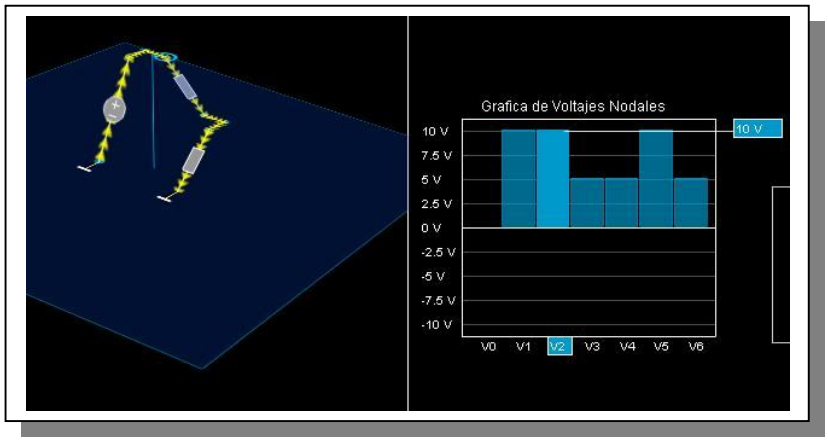


Figura I.3. Gráfica de barras interactiva y conectada con la representación del circuito en 3D.

Los resultados de la simulación del circuito en corriente alterna, también se presentan en una imagen animada e interactiva en tres dimensiones (3D). El usuario podrá "girar" la imagen para observar el circuito desde diferentes ángulos, como se muestra en la Figura I.4.

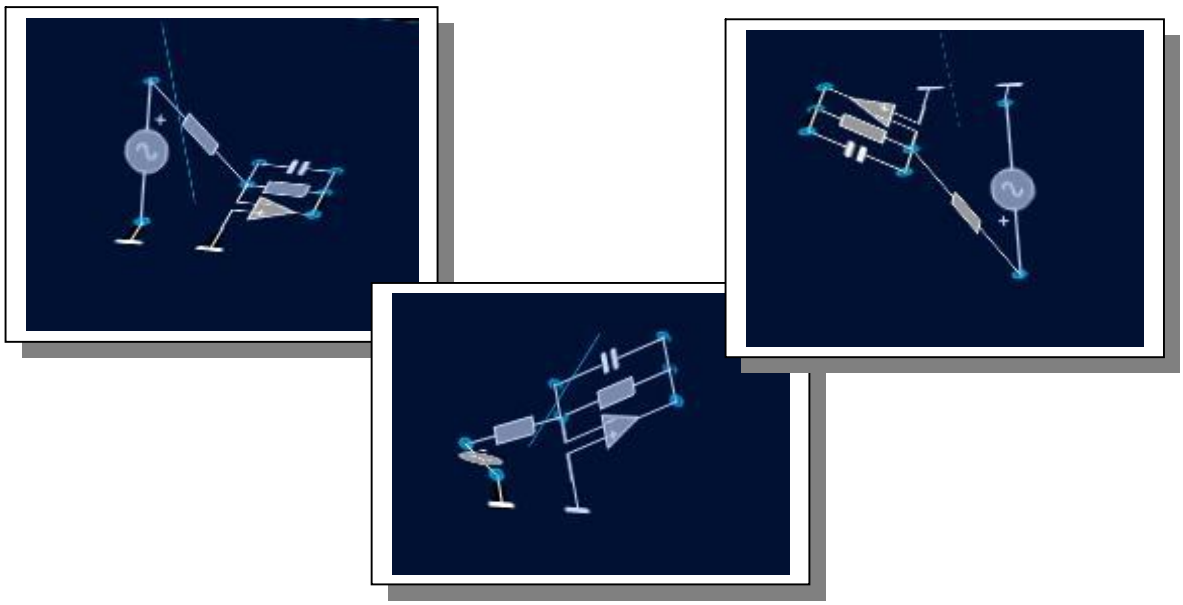


Figura I.4. Diferentes "vistas" de los resultados de la simulación en AC (magnitud y fase de los voltajes nodales) de un filtro activo pasa bajas de primer orden.

En dicha figura, se presentan varias vistas en 3D de un filtro activo pasa bajas de primer orden construido con un amplificador operacional. El plano mostrado representa nuevamente el plano de referencia de potencial "cero" volts o tierra. Las uniones entre los componentes del circuito, representan los nodos del mismo a diferentes potenciales (alturas) con respecto al plano de tierra, mientras

que el diferente movimiento de cada nodo con respecto al tiempo, representa el defasamiento respectivo.

La altura o distancia de los nodos respecto al plano de referencia es proporcional a la magnitud del voltaje nodal complejo asociado a dicho nodo, mientras que el movimiento de cada nodo con respecto al tiempo, se encuentra asociado con la fase de dicho nodo.

Además de las gráficas anteriores, en la simulación en AC se presenta la misma gráfica de barras explicada anteriormente, en la cual, ahora se representa el valor de la magnitud de los voltajes nodales complejos para cada barra. De igual forma, como ya se explicó, cuando se coloca el mouse sobre alguna barra, ésta se ilumina, se reporta el valor de dicha magnitud de voltaje y el número de nodo asignado a éste, mientras que, al mismo tiempo, se resalta mediante un círculo concéntrico, el nodo correspondiente en la gráfica en 3D.

También, en la simulación en AC, se presenta una gráfica de respuesta en frecuencia asociada al nodo seleccionado en la gráfica de barras. En la Figura I.5 se muestra un ejemplo de lo explicado anteriormente.

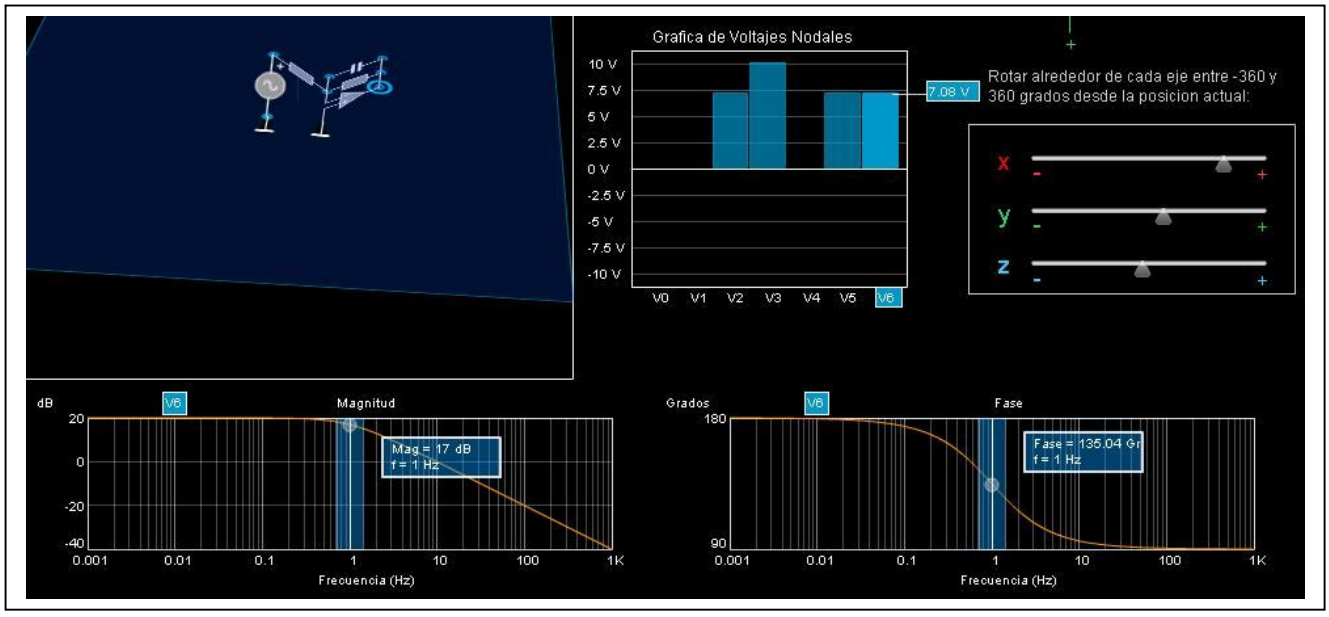


Figura I.5. Ejemplo de selección de la barra correspondiente al nodo “V6” del circuito. Obsérvese el circulo concéntrico que aparece en la gráfica en 3D, indicando la ubicación de dicho nodo seleccionado. Obsérvese también que la gráfica de Respuesta en Frecuencia mostrada en la parte inferior, es la que corresponde a dicho nodo seleccionado.

En este reporte se presenta el proyecto terminal “SIMULADOR DIDÁCTICO EN 3D DE CIRCUITOS ELÉCTRICOS Y ELECTRÓNICOS”, cuyos objetivos propuestos fueron los siguientes:

OBJETIVO GENERAL:

- Realizar un simulador didáctico en 3D de circuitos eléctricos y electrónicos, que permita analizar circuitos en CD y CA a una frecuencia determinada.

OBJETIVOS PARTICULARES:

- Mediante técnicas de Programación Orientada a Objetos, crear un sistema basado en el patrón de programación **MVC** (**M**odelo – **V**ista – **C**ontrolador) .

El reporte se divide en los siguientes capítulos

II. DESCRIPCIÓN DEL PROYECTO.

En el “SIMULADOR DIDÁCTICO EN 3D DE CIRCUITOS ELÉCTRICOS Y ELECTRÓNICOS”, se pueden realizar tres acciones:

1. Dibujar un circuito en el editor gráfico en 2D en el modo de **EDICIÓN** (default).
2. Simular el circuito dibujado en corriente directa (DC) en el modo **SIMULAR EN DC**. En este modo también se presentan de manera automática los resultados de dicha simulación en dos formatos: una **gráfica de barras** y una **gráfica dinámica en 3D**. Ambas gráficas son interactivas.
3. Simular el circuito dibujado en corriente alterna (AC) en el modo **SIMULAR EN AC**. En este modo también se presentan de manera automática los resultados de dicha simulación en tres formatos: una **gráfica de barras**, una **gráfica de respuesta en frecuencia (Bode)** y una **gráfica dinámica en 3D**. Las tres gráficas son interactivas.

La interactividad en las gráficas consiste en los siguientes procesos:

- **Gráfica de barras** – Al colocar el puntero del “mouse” sobre una “barra”, se selecciona automáticamente el “nodo” correspondiente en la gráfica dinámica en 3D y la función de respuesta en frecuencia correspondiente.
- **Gráfica de respuesta en frecuencia (Bode)** – Existe un cursor sobre esta gráfica, el cual puede desplazarse sobre la curva proporcionando los valores “x” y “y” de dicha curva.
- **Grafica dinámica en 3D** – Estas gráficas pueden girar alrededor de sus tres ejes. El giro, para seleccionar el ángulo de vista mas adecuado para el circuito, se puede realizar de dos maneras: mediante el movimiento del puntero del “mouse” sobre la misma gráfica en 3D, o bien, mediante tres “controles deslizantes” (uno para cada eje).

La información que se presenta en cada gráfica es la siguiente:

- **Gráfica de barras**:
  - En **DC**: las “barras” representa el valor del voltaje nodal de cada uno de los nodos del circuito.
  - En **AC**: las “barras” representan las magnitudes de los valores complejos de los voltajes nodales del circuito.

- **Gráfica de respuesta en frecuencia (Bode)**:
  - **Gráfica de Magnitud**: Muestra el comportamiento, como una curva continua, de la magnitud del valor complejo de los voltajes nodales del circuito en función de la frecuencia.

    - **Gráfica de Fase**: Muestra el comportamiento, como una curva continua, de la fase del valor complejo de los voltajes nodales del circuito en función de la frecuencia.

- **Graficas dinámica en 3D**:
    - En **DC**: Muestra los voltajes nodales del circuito y las corrientes de rama del mismo, en el mismo dibujo del circuito elaborado, en 3D.
    - En **AC**: Muestra las magnitudes y fases, en función del tiempo, de los voltajes nodales del circuito, en el mismo dibujo del circuito elaborado, en 3D.

La interactividad en las gráficas dinámicas en 3D consiste, como ya se mencionó, en que el usuario podrá seleccionar el “ángulo de observación” del circuito, girando éste sobre cualquier eje en el espacio 3D.

En el siguientes secciones se presentan las descripciones mas detalladas de cada uno de los modos de operación del simulador.

II.1 Modo de EDICION

El editor grafico en 2D permite dibujar un circuito eléctrico o electrónico en un área específica de la pantalla del monitor de una computadora. En la Figura II.1 se muestra la pantalla del editor, así como su estructura dividida en cinco áreas principales.

Figura II.1. Editor gráfico en 2D

Este modo de edición es el modo por default al arrancar el simulador, sin embargo, al cambiar el modo de funcionamiento a SIMULAR EN DC o SIMULAR EN AC, es posible regresar a este modo presionando el botón EDICIÓN.

A continuación se presenta una breve explicación de cada una de las áreas.

II.1.1 Menú Principal

El **menú principal** está formado por tres botones que permiten seleccionar uno de los tres estados del sistema, como se muestra en la Figura II.2

Figura II.2. Menú principal

- **EDICIÓN** –En este estado se puede **dibujar** un circuito en el **área de dibujo** y **editar** los valores de los componentes utilizados en éste.
- **SIMULAR EN DC** – Realiza la **simulación del circuito en corriente directa (DC)**, calculando los voltajes nodales y las corrientes de rama en el circuito.
- **SIMULAR EN AC** - Realiza la **simulación del circuito en corriente alterna (AC)**, calculando la magnitud y fase de los voltajes nodales .

II.1.2 Barra de Estado

En la Figura II.3 se muestra la **barra de estado**, la cual le indica al usuario el **estado** que tienen el simulador de acuerdo al botón seleccionado en el menú principal.

Figura II.3. Barra de estado

Los mensajes que pueden aparecer en dicha barra son los que se indican a continuación:

- **“>>> EDITANDO**” – Dibujando un circuito y/o editando los valores de algún componente.

- “**NO EXISTE CIRCUITO QUE SIMULAR**” – Se presionó el botón “SIMULAR EN DC” ó “SIMULAR EN AC” sin tener dibujado algún circuito en el área de dibujo.
- ”**CERRAR VENTANAS DE EDICION DE CADA COMPONENTE**” – Se encuentra abierta una o mas ventanas de edición de algun(os) componente(s).
- “**NO SE HA DEFINIDO EL NODO DE TIERRA**” – No está dibujado el símbolo de “tierra” en el área de dibujo.
- “**CIRCUITO NO CONECTADO A TIERRA**” – El circuito dibujado no tiene algún nodo conectado a “tierra”.
- “**ELEMENTOS DESCONECTADOS DEL CIRCUITO**” – Existe uno o mas elementos que no están conectados entre sí.

- "**FUENTES EN CONEXION NO VALIDA**" – Fuente de voltaje en "corto circuito" ó fuente de corriente en "circuito abierto".

- "**SIMULANDO EN DC**" – Se presionó el botón "SIMULAR EN DC".
- "**SIMULACION VALIDA EN > DC <**" – Los resultados de la simulación en DC son válidos.
- "**SIMULANDO EN AC**" - Se presionó el botón "SIMULAR EN AC".
- "**SIMULACION VALIDA EN > AC** <" - Los resultados de la simulación en AC son válidos.
- "**PROCESANDO . . .**" – Se encuentra en proceso de simulación en AC.

- "**01:: ERROR EN LA SIMULACION: REVISAR EL CIRCUITO**" – Un voltaje nodal no pudo ser calculado.
- "**02:: ERROR EN LA SIMULACION: REVISAR EL CIRCUITO**" – Una corriente de rama no pudo ser calculada.
- "**03:: ERROR EN LA SIMULACION: REVISAR EL CIRCUITO**" - Un valor de magnitud y fase de algún voltaje nodal no pudo ser calculado.

II.1.3 Area de Dibujo

En el **Area de Dibujo** se realiza el dibujo del circuito que se va a simular, generando dicho dibujo mediante la interconexión de los elementos seleccionados en el **Menú de Componentes**, o bien, seleccionando algún circuito, previamente definido, del **Menú de Circuitos**, como se muestra en la Figura II.4.

La selección, tanto de los componentes como de los circuitos predefinidos, se realiza presionando los botones correspondientes que aparecen en sus respectivos menús.

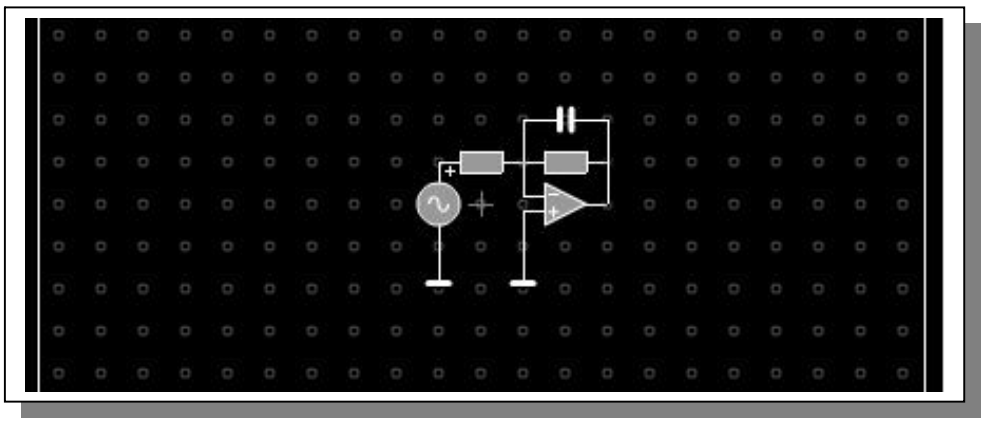


Figura II.4. Ejemplo de un circuito dibujado en el **Area de Dibujo** del editor de circuitos

En esta área de dibujo, se puede también editar los valores de los componentes que conforman el circuito bajo prueba. Dicha edición se realiza

abriendo las ventanas de edición de cada componente y escribiendo, en dicha ventana, los valores requeridos por el usuario, como se puede observar en la Figura II.5.

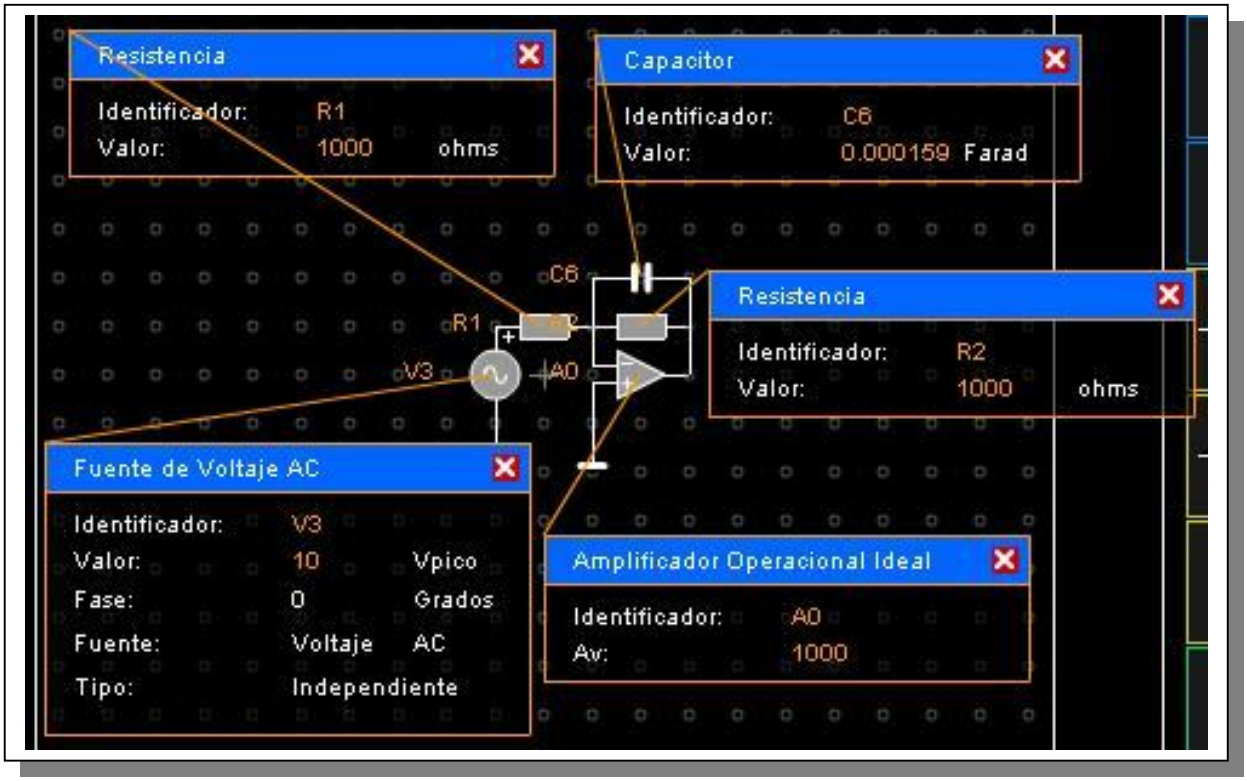


Figura II.4. Ventanas de edición de los valores de los componentes de un circuito dibujado en el **Area de Dibujo** del editor de circuitos

II.1.4 Menú de Componentes

El **Menú de Componentes** consiste en una colección de símbolos (botones) de componentes eléctricos y electrónicos, con diferentes orientaciones, que podrán ser seleccionados por el usuario para dibujar un determinado circuito en el **Area de Dibujo**. En la Figura II.5 se presenta la pantalla correspondiente a este menú.

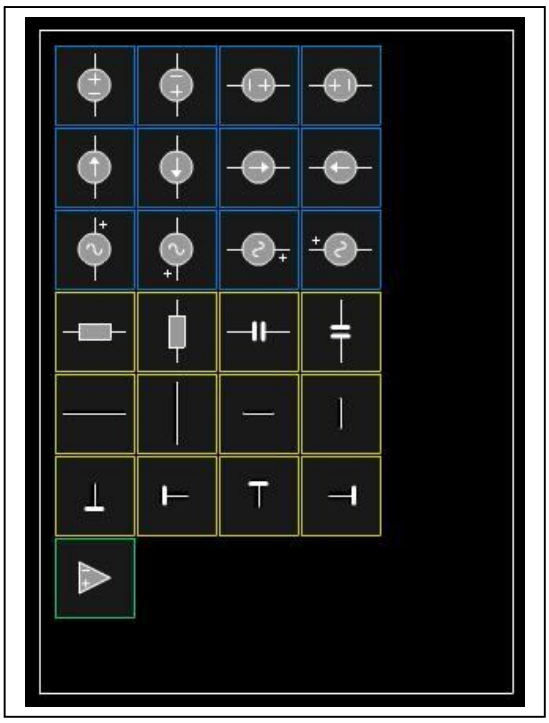


Figura II.5. Menú de Componentes

Los componentes disponibles en este menú son los siguientes:

- Fuentes de Voltaje de DC
- Fuentes de Corriente de DC
- Fuentes de Voltaje de AC
- Resistencias
- Capacitores
- Alambres “largo” y “corto”
- Tierra
- Amplificador Operacional

Cada componente, con su respectiva orientación, constituye un botón que, al presionarlo, lanza el componente respectivo al Area de Dibujo del editor. Una vez ahí, el símbolo del componente se puede trasladar colocando el mouse sobre el símbolo, presionar el botón izquierdo del mouse y arrastrarlo a la posición deseada. Aplicando un “doble clic” sobre el símbolo del componente, se abre una

ventana de edición del componente, en donde se puede editar o cambiar algunos valores de éste.

II.1.5 Menú de Circuitos

El **Menú de Circuitos** consiste en una colección de circuitos eléctricos y electrónicos (botones), que podrán ser seleccionados por el usuario para lanzar un determinado circuito al **Area de Dibujo**. En la Figura II.6 se presenta la pantalla correspondiente a este menú.

Figura II.6. Menú de Circuitos

Los circuitos disponibles en este menú, al momento de la elaboración de este reporte, son los siguientes:

- Divisor de voltaje resistivo
- Filtro Activo Pasa Bajas de 1er orden

Al igual que en el menú de componentes, al presionar un botón se lanza el circuito respectivo al Area de Dibujo del editor. Una vez ahí, el circuito se puede simular, alterar, redibujar o editar como el usuario lo requiera.

II.2 Modo de SIMULACION EN DC

Cuando el usuario presiona el botón **SIMULAR EN DC**, el simulador analiza el circuito dibujado en el Area de Dibujo. Si existe algún problema con dicho dibujo, el simulador lo reportará al usuario a través de la **Barra de Estado**, informándole del posible error existente en el circuito. Una vez corregido el dibujo de éste, al presionar nuevamente el botón **SIMULAR EN DC**, el simulador calcula todos los voltajes nodales y las corrientes de rama del circuito, desplegando automáticamente los resultados en dos formatos: una **gráfica de barras** y una **gráfica dinámica en 3D**, como se muestra en la Figura II.7

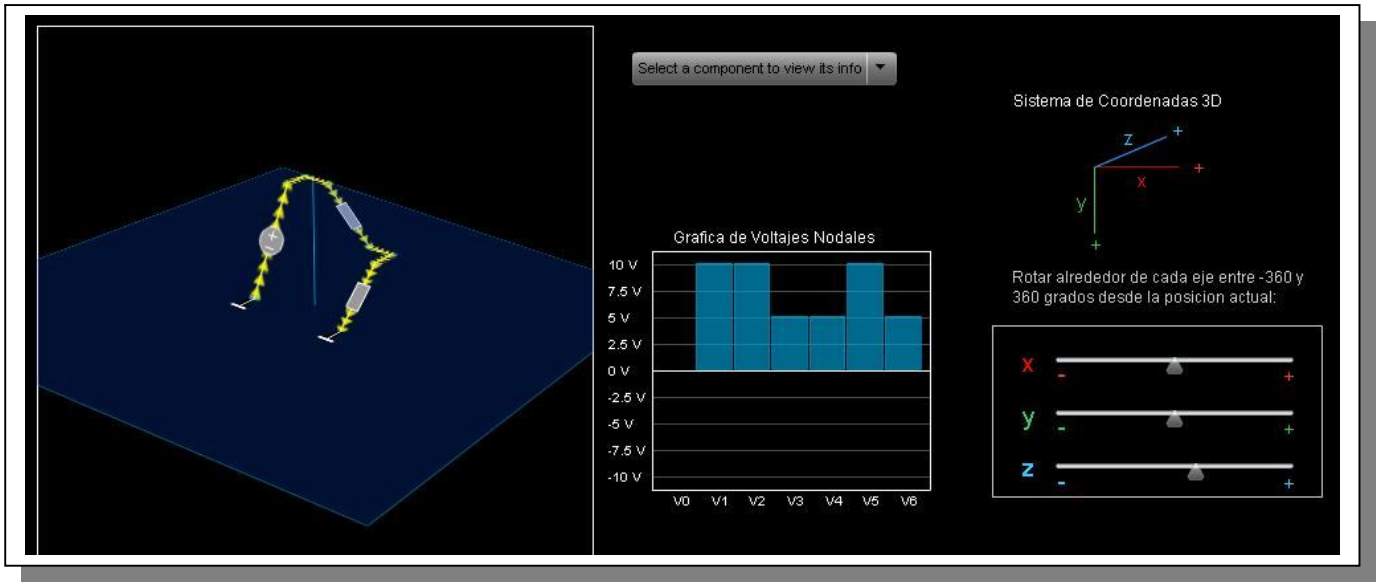


Figura II.7. Simulación de un circuito en DC. Se observa a la izquierda la gráfica dinámica en 3D, al centro la gráfica de barras y a la derecha los controles deslizantes para girar la gráfica en sus tres ejes

La **gráfica dinámica en 3D** presenta los resultados de la simulación en DC como se explica a continuación. Los voltajes nodales están representados por la "altura" de éstos al plano de potencial "0 volts", el cual constituye el plano de referencia para todos los nodos y está representado en la gráfica con el cuadrado sobre el cual está realzado el circuito. Las corrientes de cada rama están representadas por un "flujo de flechas", es decir, son flechas que se mueven en el sentido convencional de la corriente: de potenciales mayores (alturas mayores) a potenciales menores (alturas menores), cuyo ancho corresponde, de manera representativa, con el valor de la corriente de rama para cada componente.

Esta gráfica es interactiva en el sentido de que puede "girarse" en cualquiera de sus tres ejes x-y-z, colocando el puntero del mouse sobre el Area de Dibujo, presionando el botón izquierdo del mouse, y arrastrando éste hasta lograr el giro deseado. Otra forma de girar esta grafica consiste en utilizar los controles deslizantes que aparecen en el extremo derecho de la pantalla. Cada uno de ellos permite el giro de la gráfica alrededor del eje indicado en dicho control.

En la Figura II.8 se muestran tres ángulos diferentes de giro de la gráfica en 3D para un circuito divisor de voltaje resistivo, simulado en DC.

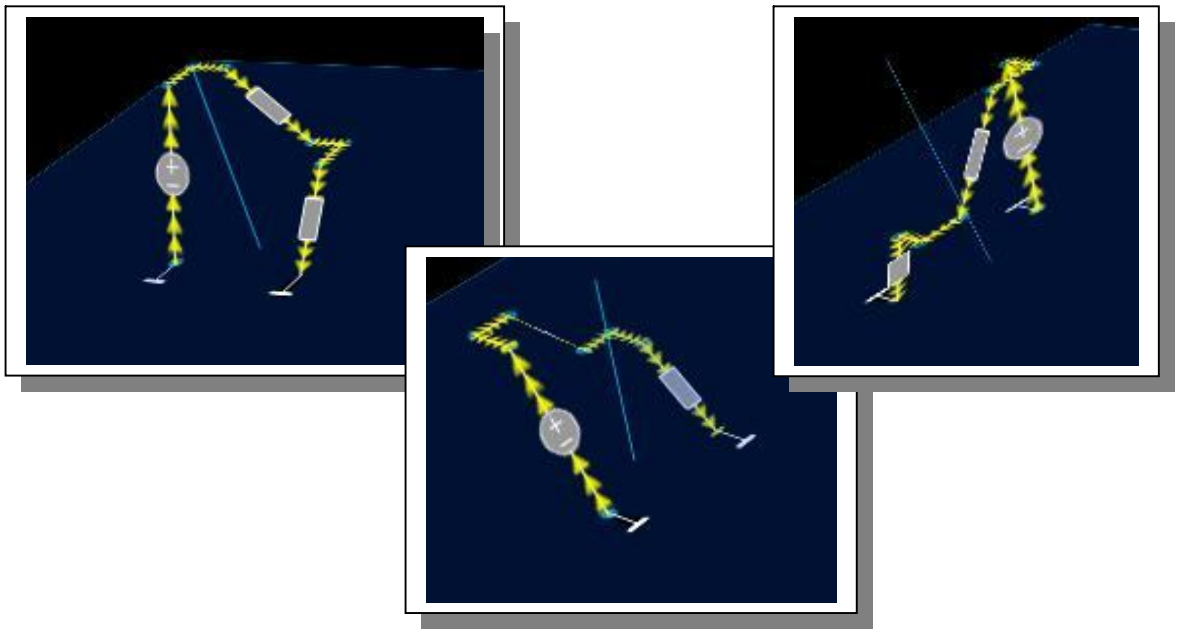


Figura II.8. Diferentes “vistas” de los resultados de la simulación en DC (voltajes nodales y corrientes de rama) de un circuito divisor de voltaje resistivo.

La **gráfica de barras** muestra los voltajes nodales de cada nodo del circuito. El identificador de cada nodo aparece en el eje horizontal de la gráfica, mientras que en el eje vertical aparece la acotación, en voltaje, correspondiente al circuito que se muestra en la grafica en 3D. La altura de cada barra representa el voltaje nodal correspondiente a cada nodo.

Esta gráfica también es interactiva, en el sentido de que al colocar el puntero del mouse “sobre” alguna barra, ésta se ilumina y, además de mostrar el valor numérico del voltaje nodal correspondiente (la altura de la barra), en la gráfica del circuito en 3D, aparece un circulo dinámico concéntrico alrededor del nodo correspondiente, indicando de este modo al usuario, la localización, en el circuito, del nodo correspondiente a la barra que se selecciona.

En la Figura II.9 se muestra un ejemplo en el que se ha seleccionado la barra correspondiente al nodo “V2”. Obsérvese como dicha barra se ilumina, presenta el valor numérico del voltaje de ese nodo y, además, aparece un círculo concéntrico en uno de los nodos de la parte superior izquierda de la gráfica en 3D.

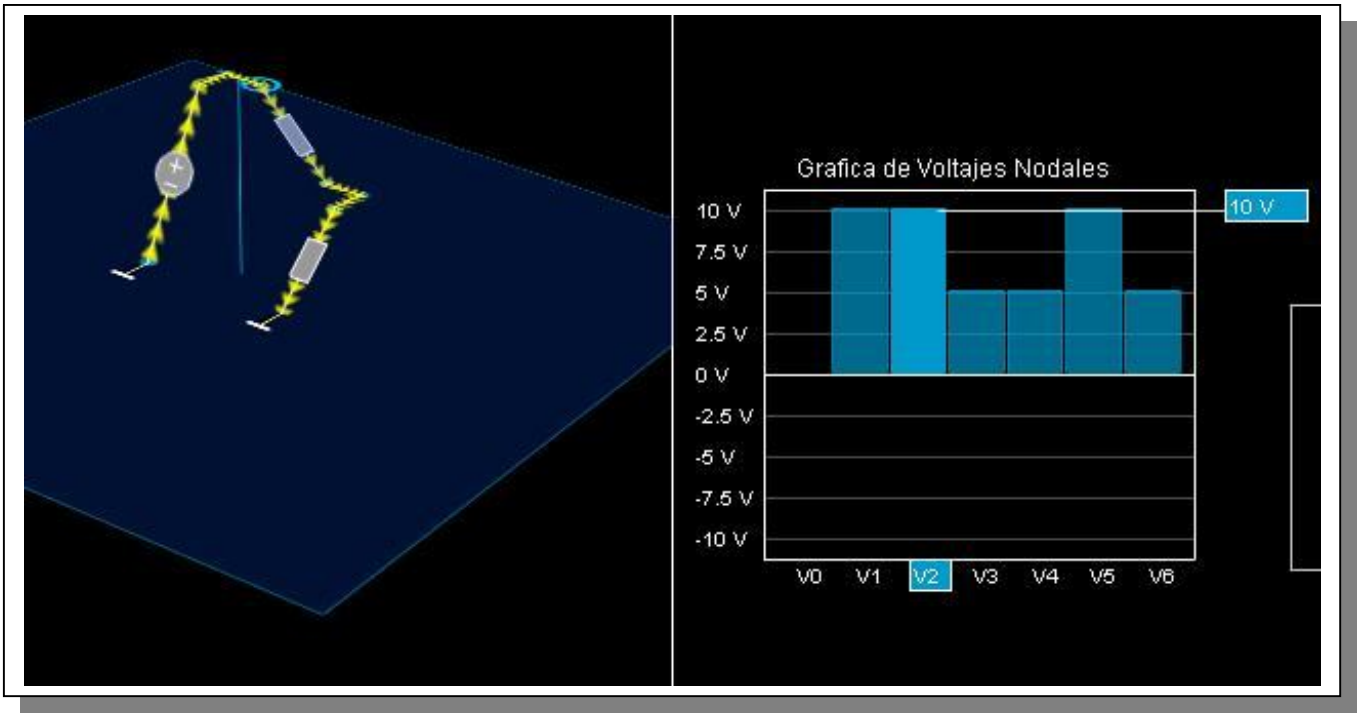


Figura II.9. Ejemplo de selección de la barra correspondiente al nodo “V2” del circuito. Obsérvese el circulo concéntrico que aparece en la gráfica en 3D, indicando la ubicación de dicho nodo seleccionado.

II.3 Modo de SIMULACION EN AC

De manera similar al modo anterior, cuando el usuario presiona el botón **SIMULAR EN AC**, el simulador analiza el circuito dibujado en el Area de Dibujo. Si existe algún problema con dicho dibujo, el simulador lo reportará al usuario a través de la **Barra de Estado**, informándole del posible error existente en el circuito. Una vez corregido el dibujo de éste, al presionar nuevamente el botón **SIMULAR EN AC**, el simulador calcula todos los valores complejos de los voltajes

nodales del circuito, considerando a todas la fuentes de voltaje “senoidal” que estén conectadas al circuito, desplegando automáticamente los resultados en tres formatos: una **gráfica de respuesta en frecuencia (Bode)**, una **gráfica de barras** y una **gráfica dinámica en 3D**, como se muestra en la Figura II.10

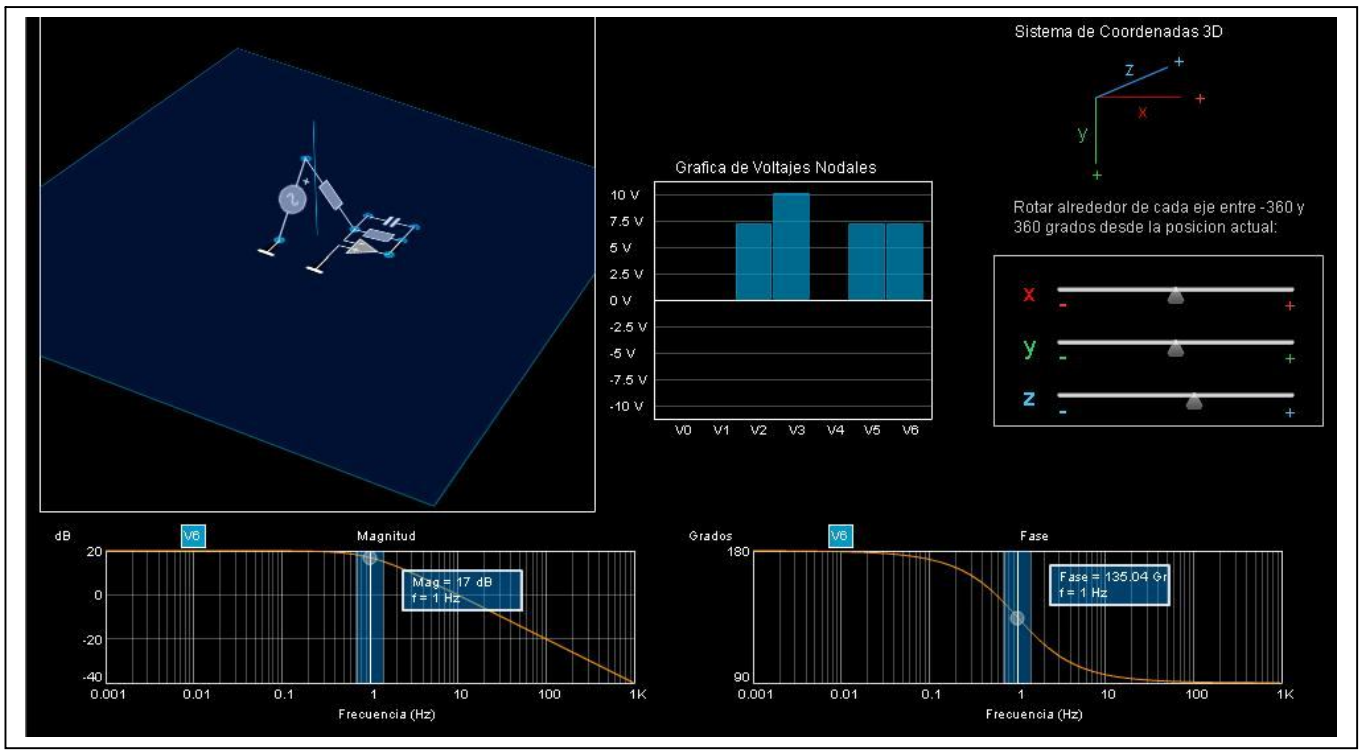


Figura II.10. Simulación de un circuito en AC. Se observa a la izquierda la gráfica dinámica en 3D, al centro la gráfica de barras y a la derecha los controles deslizantes para girar la gráfica en sus tres ejes. El parte inferior se encuentra la gráfica de Respuesta en Frecuencia (Bode): magnitud y fase.

La **gráfica dinámica en 3D** presenta los resultados de la simulación en AC como se explica a continuación. En este gráfica se hace uso de la variable “tiempo” para representar el comportamiento dinámico del circuito. La excitación del circuito lo constituyen las fuentes de voltaje senoidales conectadas a éste. La respuesta del circuito a dicha excitación, se representa con el “movimiento” de las alturas de los nodos respecto al plano de potencial “0 volts”, el cual constituye el plano de referencia para todos los nodos y está representado en la gráfica con el cuadrado sobre el cual está oscilando el circuito. Al utilizar el tiempo como variable de esta gráfica, se pueden observar fácilmente los diferentes defasamientos que presentan los nodos del circuito entre si.

Esta gráfica es interactiva en el sentido de que puede “girarse” en cualquiera de sus tres ejes x-y-z, colocando el puntero del mouse sobre el Area de Dibujo, presionando el botón izquierdo del mouse, y arrastrando éste hasta lograr el giro deseado. Otra forma de girar esta grafica consiste en utilizar los controles

deslizantes que aparecen en el extremo derecho de la pantalla. Cada uno de ellos permite el giro de la gráfica alrededor del eje indicado en dicho control.

En la Figura II.11 se muestran tres ángulos diferentes de giro de la gráfica en 3D para un circuito que funciona como filtro activo pasa bajas de primer orden, simulado en AC.

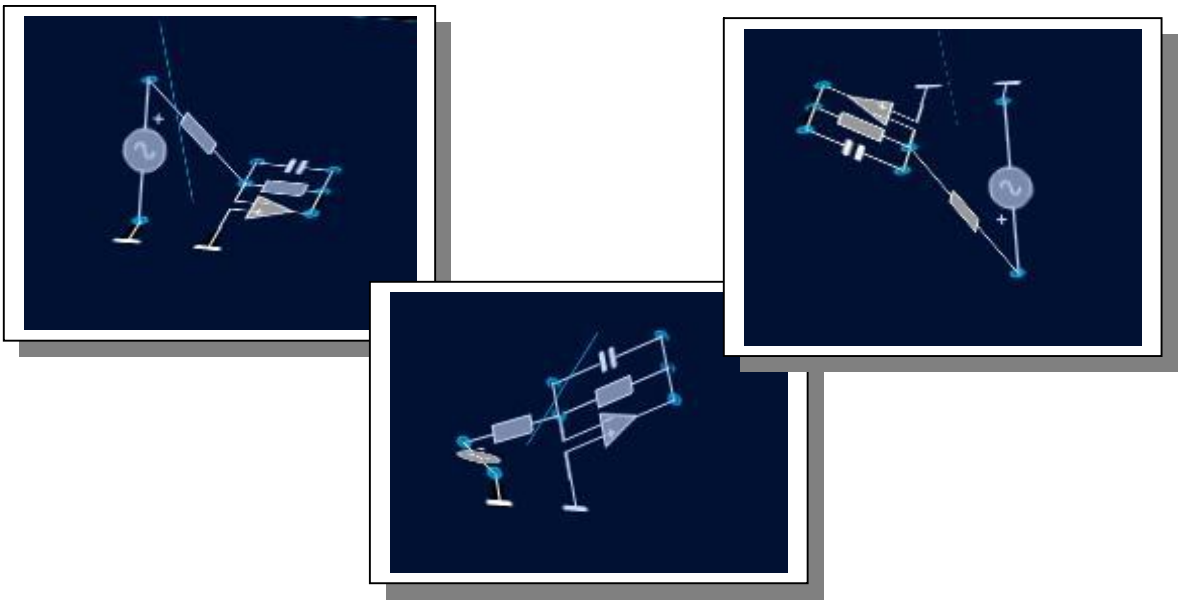


Figura II.11. Diferentes "vistas" de los resultados de la simulación en AC (magnitud y fase de los voltajes nodales) de un filtro activo pasa bajas de primer orden.

La **gráfica de barras** muestra únicamente las magnitudes de los voltajes nodales de cada nodo del circuito. El identificador de cada nodo aparece en el eje horizontal de la gráfica, mientras que en el eje vertical aparece la acotación, en voltaje, correspondiente al circuito que se muestra en la grafica en 3D. La altura de cada barra representa la magnitud del voltaje nodal correspondiente a cada nodo.

Esta gráfica también es interactiva, en el sentido de que al colocar el puntero del mouse "sobre" alguna barra, ésta se ilumina y, además de mostrar el valor numérico de la magnitud del voltaje nodal correspondiente (la altura de la barra), en la gráfica del circuito en 3D, aparece un circulo dinámico concéntrico alrededor del nodo correspondiente, indicando de este modo al usuario, la localización, en el circuito, del nodo correspondiente a la barra que se selecciona.

En la Figura II.12 se muestra un ejemplo en el que se ha seleccionado la barra correspondiente al nodo "V6". Obsérvese como dicha barra se ilumina, presentando el valor numérico de la magnitud del voltaje de ese nodo y, además, aparece un círculo concéntrico en uno de los nodos de la parte superior derecha de la gráfica en 3D.

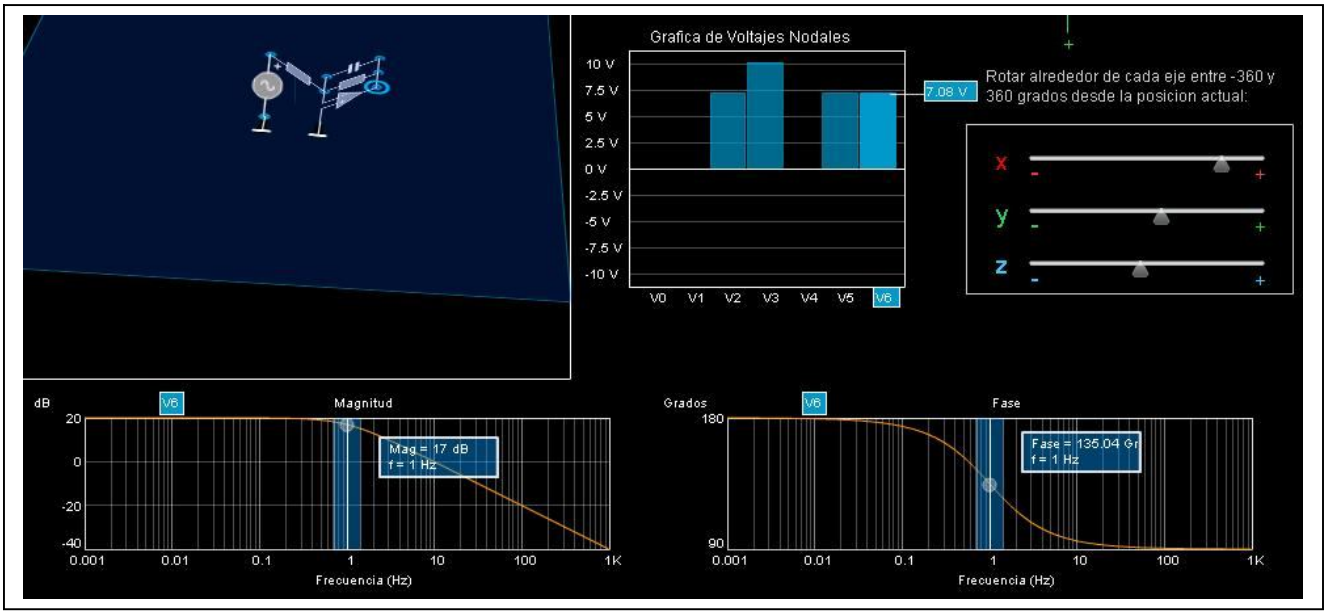


Figura II.12. Ejemplo de selección de la barra correspondiente al nodo "V6" del circuito. Obsérvese el circulo concéntrico que aparece en la gráfica en 3D, indicando la ubicación de dicho nodo seleccionado. Obsérvese también que la gráfica de Respuesta en Frecuencia mostrada en la parte inferior, es la que corresponde a dicho nodo seleccionado.

La **gráfica de respuesta en frecuencia (Bode)**, muestra la magnitud y la fase de los voltajes nodales de cada nodo del circuito. El identificador de cada nodo aparece en la parte superior izquierda, tanto de la gráfica de magnitud como de la gráfica de fase. En la parte inferior de ambas gráficas, se presenta la acotación del eje de frecuencia (eje horizontal) en "Hertz" y en una escala logarítmica, mientras que en el eje vertical se presenta una acotación en "dB" para la gráfica de magnitud y en "grados" para la gráfica de la fase.

Esta gráfica también es interactiva, en el sentido de que al colocar el puntero del mouse "sobre" el cursor de cualquiera de las gráficas, éste se podrá desplazar al mover el mouse teniendo presionado su botón izquierdo. Al moverse el cursor, irá apareciendo en el cuadro correspondiente, el valor tanto de la frecuencia como de la magnitud y la fase respectiva.

En la Figura II.13 se presentan dos ejemplos en los cuales el cursor de la gráfica de respuesta en frecuencia, se presenta en dos posiciones diferentes.

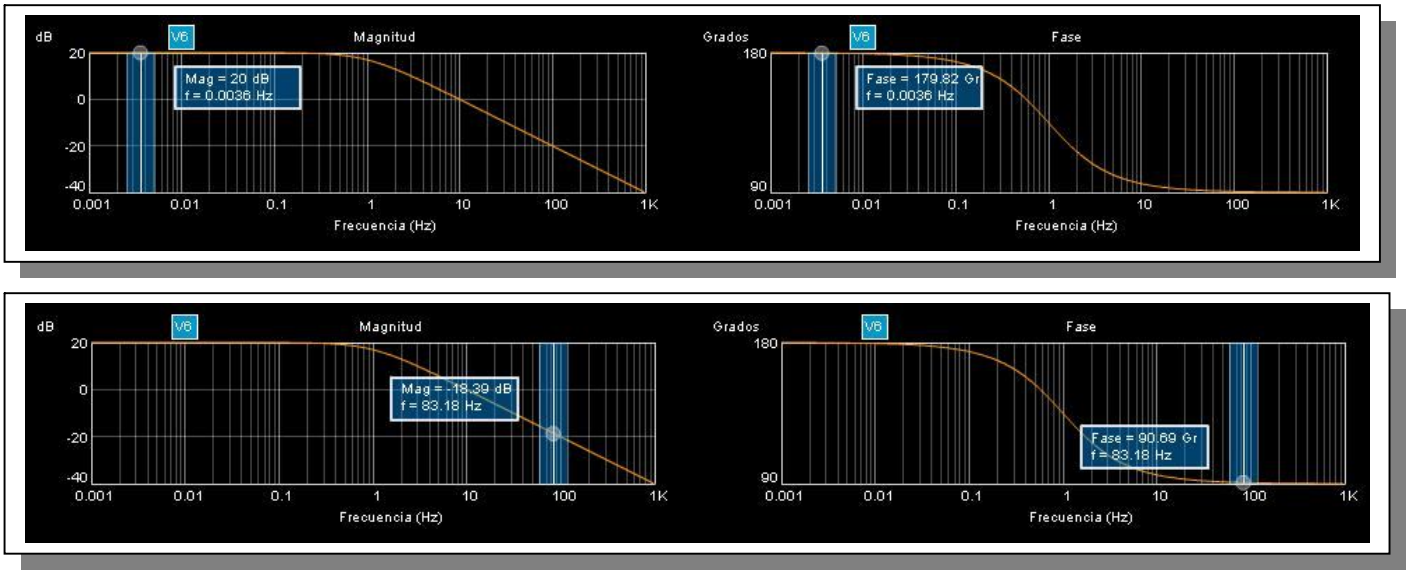


Figura II.13. Ejemplo de selección de dos diferentes puntos de la grafica de Respuesta en Frecuencia mediante el movimiento del cursor

III. ESTRUCTURA DEL PROYECTO.

Las tres funciones básica que tiene el proyecto son:

1. Permitir dibujar un circuito eléctrico o electrónico en un “editor” de circuitos.
2. Simular el circuito en corriente directa (CD) y en corriente alterna (CA).
3. Mostrar los resultados de la simulación en diferentes formatos de gráficas: gráfica de barras, gráfica de línea continua y gráfica dinámica en 3D.

Para la elaboración del proyecto se eligió un lenguaje de programación orientado a objetos, por todas las características inherentes a éste, como la abstracción, el encapsulamiento, la herencia y el polimorfismo, las cuales facilitaron la elaboración del mismo. Sin embargo, uno de los puntos mas difíciles en el desarrollo del proyecto, fue la elección de cómo estructurar el proyecto, que “forma” darle, que objetos crear y como conectarlos entre sí para construir el editor, el simulador, las representaciones en 3D y todos los detalles que se fueron presentando, como la integración de todos los objetos para que funcionaran sincronizados y en armonía.

La elección de la estructura se basó en los objetivos del campo de la programación conocido como “diseño de patrones”: El “diseño de patrones” es un conjunto de herramientas que permite hacer frente a los constantes “cambios” en el diseño de software, permitiendo al programador re-usar la mayoría del software ya desarrollado, y realizar los cambios necesarios sin alterar el código ya existente, es decir, permite un alto grado de ”flexibilidad” en la renovación de las interconexiones entre los objetos ya existentes y los objetos nuevos necesarios para realizar el cambio o actualización requerido en la aplicación. En el caso del simulador que se desarrolló, estos “cambios” se refieren a la adición de nuevos componentes que no se encuentran actualmente en el proyecto.

Con este criterio, se eligió un patrón de programación llamado Modelo-Vista-Controlador (MVC), como la estructura base principal sobre la cual se comenzó a construir el proyecto. Este patrón se eligió por dos razones: 1) por ser el más utilizado en el desarrollo de la instrumentación virtual (instrumentos analógicos y digitales desarrollados alrededor de un procesador o computadora) y simuladores (por ejemplo simuladores de vuelo); y 2) por ser el que más se adapta a las necesidades del proyecto.

Durante el desarrollo del proyecto se vió la necesidad de incorporar también otros dos patrones de programación mas: 1) el llamado “método de la fabrica” para el diseño del editor de circuitos; y 2) el patrón conocido como “composición” para el diseño de las diferentes “vistas” en la presentación visual de los resultados (por ejemplo la gráfica de barras y las gráficas dinámicas en 3D).

Como se explicará mas adelante, existen otros patrones de programación incorporados (en forma traslapada) a las estructuras descritas en este capítulo. Para mayor claridad en la descripción de la estructura del proyecto, dichos patrones de programación se mencionarán y explicarán en los siguientes capítulos.

En este capítulo se describe la estructura del simulador en 3D, basada en los tres patrones de programación mencionados anteriormente.

III.1 Lenguaje de Programación

Aunque en la propuesta de este proyecto, se eligió como lenguaje de programación **ActionScript 3.0 para Flash (CS3)**, se optó por cambiar a **ActionScript 3.0 para Flash (CS4)**, debido a la mayor facilidad y flexibilidad que tiene **Flash CS4** para manejar objetos en 3D, así como el manejo de eventos de diferentes tipos, lo que permite crear ambientes interactivos muy variados. Además, es posible instalar las aplicaciones generadas, ya sea en servidores para su acceso remoto a través de internet, o en máquinas individuales para su acceso en forma local. El único requisito para accesar dichas aplicaciones, es tener instalado el Flash Player 10.0 o superior.

III.2 Estructura Básica: Modelo-Vista-Controlador (MVC)

La estructura básica del proyecto se basa en el patrón de programación **Modelo-Vista-Controlador** (**MVC**), cuyo diagrama a bloques se muestra en la Figura III.1.

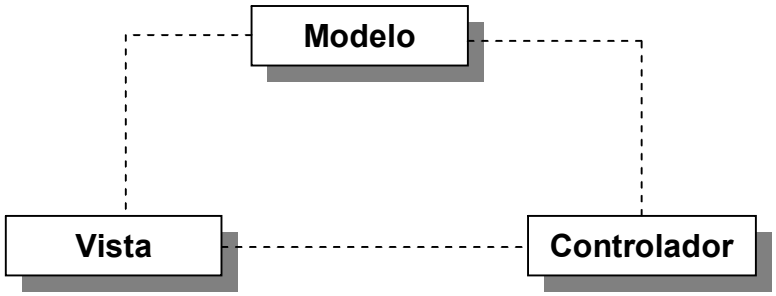


Figura III.1. Patrón de programación Modelo-Vista-Controlador (MVC)

Cada módulo representa una **clase** de la cual se obtiene un **objeto** al "instanciar" dicha **clase**, así que el diagrama a bloques de la Figura II.1 representa también la "conexión" que se establece entre los **objetos** creados a partir del mismo diagrama de **clases**. A partir de aquí, se usará indistintamente el término "bloque" o "módulo" para referirse ya sea a una **clase** o a un **objeto**.

El patrón **MVC**, como se observa de la figura, consiste de tres bloques:

- **Modelo** - Contiene el modelo matemático del circuito y todos los datos necesarios para la simulación de éste, así como la lógica que maneja el estado de todo el proyecto.
- **Vista** - Presenta al usuario los resultados de la simulación, en el monitor de la computadora, con diferentes formatos: grafica de barras, gráfica de respuesta en frecuencia y grafica dinámica en 3D.
- **Controlador** - Maneja los botones que activa el usuario para cambiar el estado de la aplicación: edición, simulación en DC y simulación en AC.

La ventaja de este modelo radica en la separación de "responsabilidades" en cada uno de los bloques, sin que ninguno de ellos se interfiera. Además, se observa que satisface completamente la funcionalidad requerida por el proyecto.

III.3 Primera Estructura Auxiliar: Patrón del Método de la Fábrica (FMP)

El nombre de este patrón de programación **Método de la Fábrica**, describe muy bien la función de esta estructura, pues efectivamente se encarga de **fabricar** diferentes **objetos** a través de una **línea de producción** previamente definida.

En el proyecto, este patrón se aplicó en el diseño del **editor**. En éste, el usuario solicita, al presionar un botón, la fabricación de un determinado componente o circuito eléctrico o electrónico (**objeto**) que será utilizado para dibujar (construir) un circuito.

El diagrama a bloques del patrón de programación **Método de la Fábrica** (**FMP**), se muestra en la Figura III.2.

Figura III.2. . Patrón de programación Método de la Fábrica (FMP).

Como se observa en la figura, el **FMP** consiste de tres bloques (dos de ellos son un grupo de bloques cada uno).

- **Cliente** - Contiene un grupo de “botones” (Menú de Componentes o Menú de Circuitos) con los cuales el usuario solicita la fabricación de un determinado componente eléctrico o electrónico, o un circuito completo.
- **Método de la Fábrica** - Fabrica con capacidad de producir componentes eléctricos y electrónicos de una terminal, dos terminales y tres terminales, o un circuito completo.
- **Producto Concreto** – Objeto con las propiedades específicas del componente o circuito fabricado

Este patrón de programación tiene la característica y ventaja, de **desacoplar** la relación tan “rígida” que se pudiera presentar entre el **cliente** y el **producto**, si la fábrica tuviera una **línea de producción** fija para producir determinados productos. El patrón de programación del **Método de la Fábrica**, permite integrar la producción de nuevos componentes de 1, 2 ó 3 terminales, sin necesidad de alterar el código de programación de la **línea de producción** de la fábrica. Unicamente se **agrega**, al código ya desarrolado de la fábrica, la definición, especificaciones y dibujo del nuevo elemento que se va a producir. Esto hace muy flexible y adaptable a la fábrica, pues no se tendrá que modificar el código si se quiere que un nuevo componente esté disponible en el editor de circuitos.

Cabe aclarar que, aunque en el FMP se pudo lograr la característica mencionada en el párrafo anterior, existe actualmente en el proyecto un objeto que no lo permite: el simulador (DC y AC). Este bloque si tiene que modificar su código para aceptar nuevos componentes.

III.4 Segunda Estructura Auxiliar: Patrón de Composición (CP)

El **Patrón de Composición** es similar a un “móvil” colgado del techo de una habitación. Si con la mano movemos algún elemento, dicho movimiento afectará a todos los demás.

Este patrón se aplicó en el diseño del módulo **Vista** de la estructura básica **MVC**, es decir, este patrón se encuentra inmerso en el módulo **Vista**. Cuando se toca con el “mouse” una barra de la gráfica de barras mostrada en este módulo, las demás gráficas (gráfica de respuesta en frecuencia y gráfica dinámica en 3D), responderán a dicho evento en diferentes formas. Esta es la analogía con el “móvil” colgado del techo, y de cómo el movimiento se transmite a los demás elementos. Este método es el que permite la interactividad con las gráficas en el bloque **Vista**.

El diagrama a bloques del patrón de programación **Composición** (**CP**), se muestra en la Figura III.3.

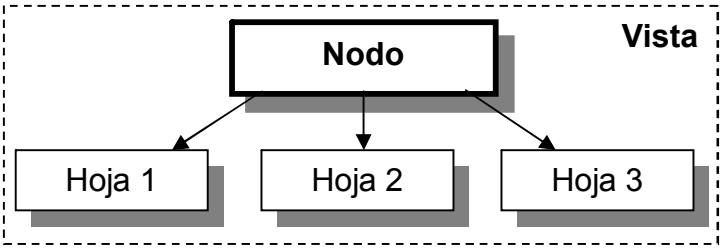


Figura III.3. Patrón de programación Composición (CP).

El patrón de **Composición** tiene dos bloques (uno de ellos es un grupo de bloques).

- **Nodo** - Constituye el manejador de las graficas (hojas), enviándoles la información que éstas requieren, y sirviendo de "puente" entre ellas mismas cuando sucede un "evento" en alguna de las hojas.
- **Hoja** – Gráfica de barras, gráfica de respuesta en frecuencia y/o gráfica dinámica en 3D (DC o AC).

III.5 Diagrama a bloques del proyecto

En la Figura III.4 se presenta un diagrama a bloques completo del SIMULADOR DIDÁCTICO EN 3D DE CIRCUITOS ELÉCTRICOS Y ELECTRÓNICOS, en donde se muestra la ubicación de los tres patrones de programación utilizados en el proyecto: **MVC**, **FMP** y **CP**.

El **MVC** constituye la base del proyecto. Su objetivo es el manejo, coordinación y control de todos los elementos del simulador.

Como ya se mencionó, el **FMP** se utiliza en el editor para manejar tanto el **Menú de Componentes** como el **Menú de Circuitos** y el **Area de Dibujo**. Su objetivo es fabricar los componentes o circuitos con las especificaciones particulares de cada uno de ellos.

El **CP** se encarga del manejo visual de los resultados de la simulación

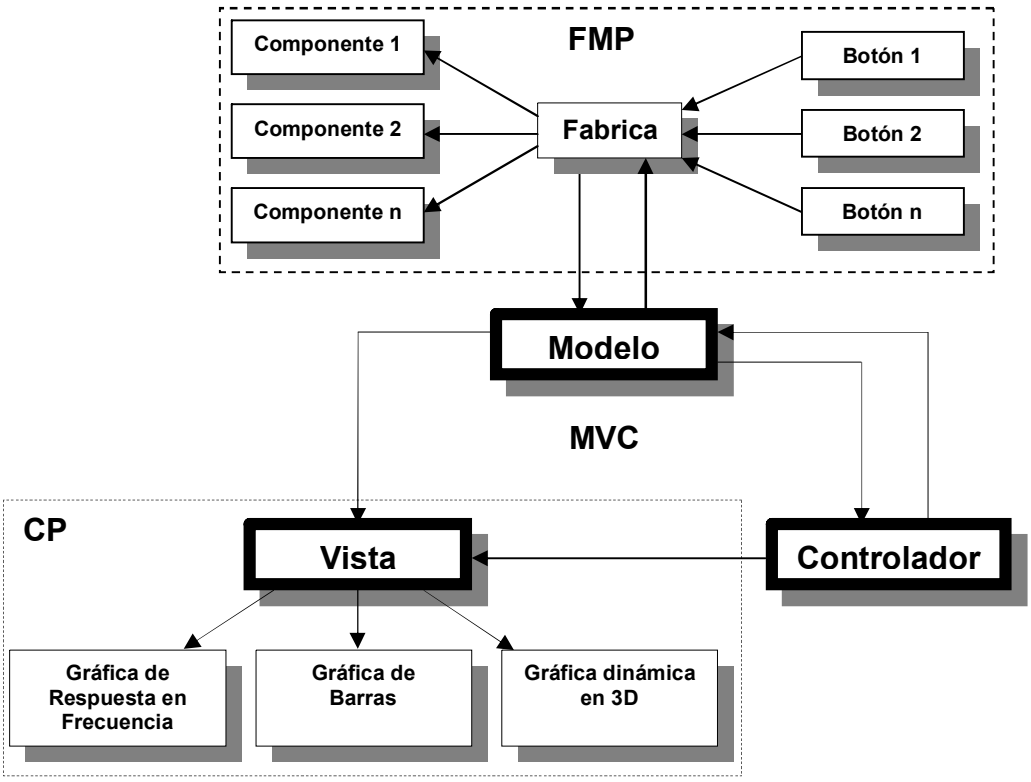


Figura III.4. Diagrama a bloques de la estructura del proyecto (integración de los patrones de programación: MVC, FMP y CP).

La relación de los patrones de programación en cada una de las pantallas que observa el usuario es la que se muestra en las Figuras III.5 a la III.8

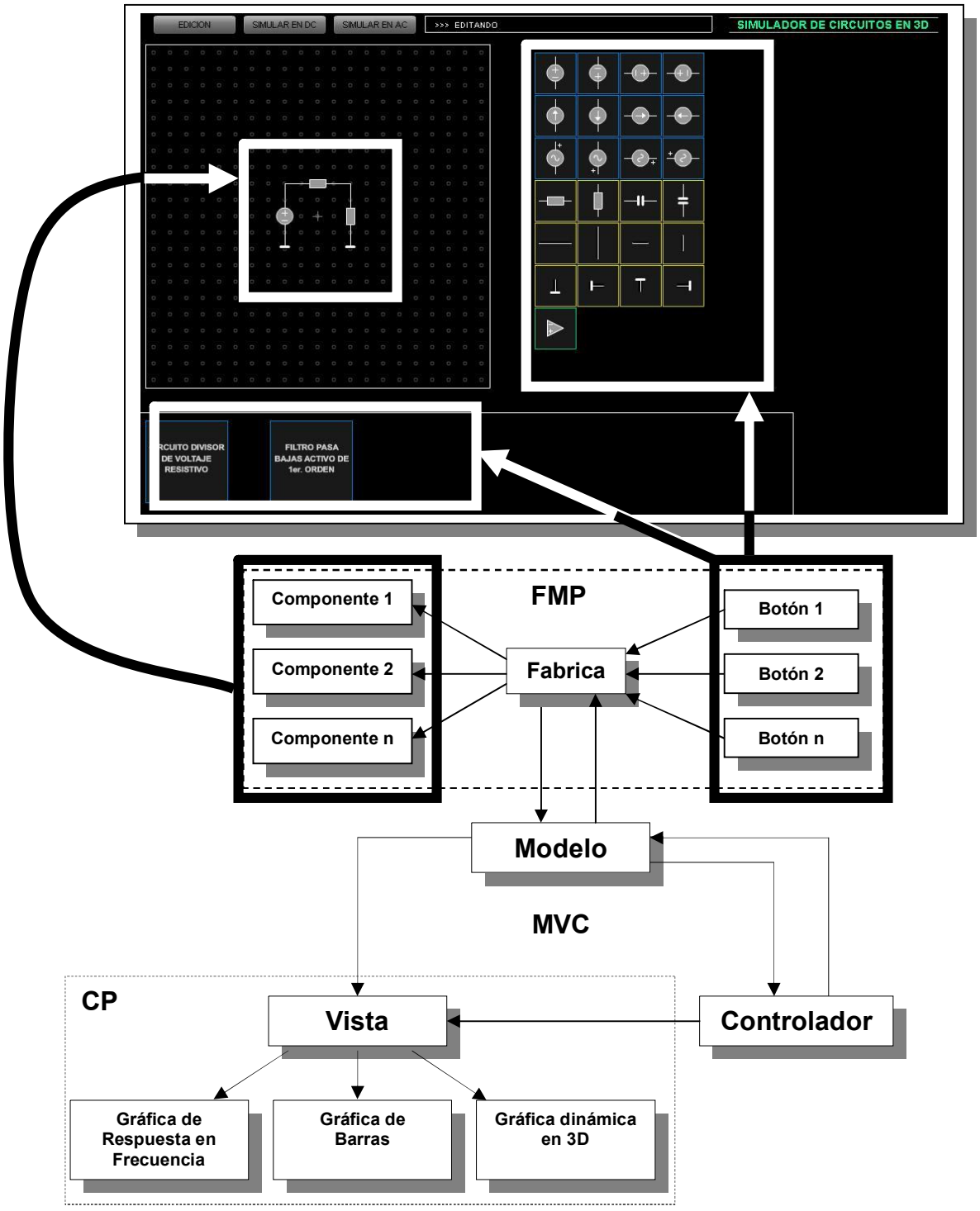


Figura III.5. Relación del patrón de programación **FMP** con la pantalla del modo de **EDICION**

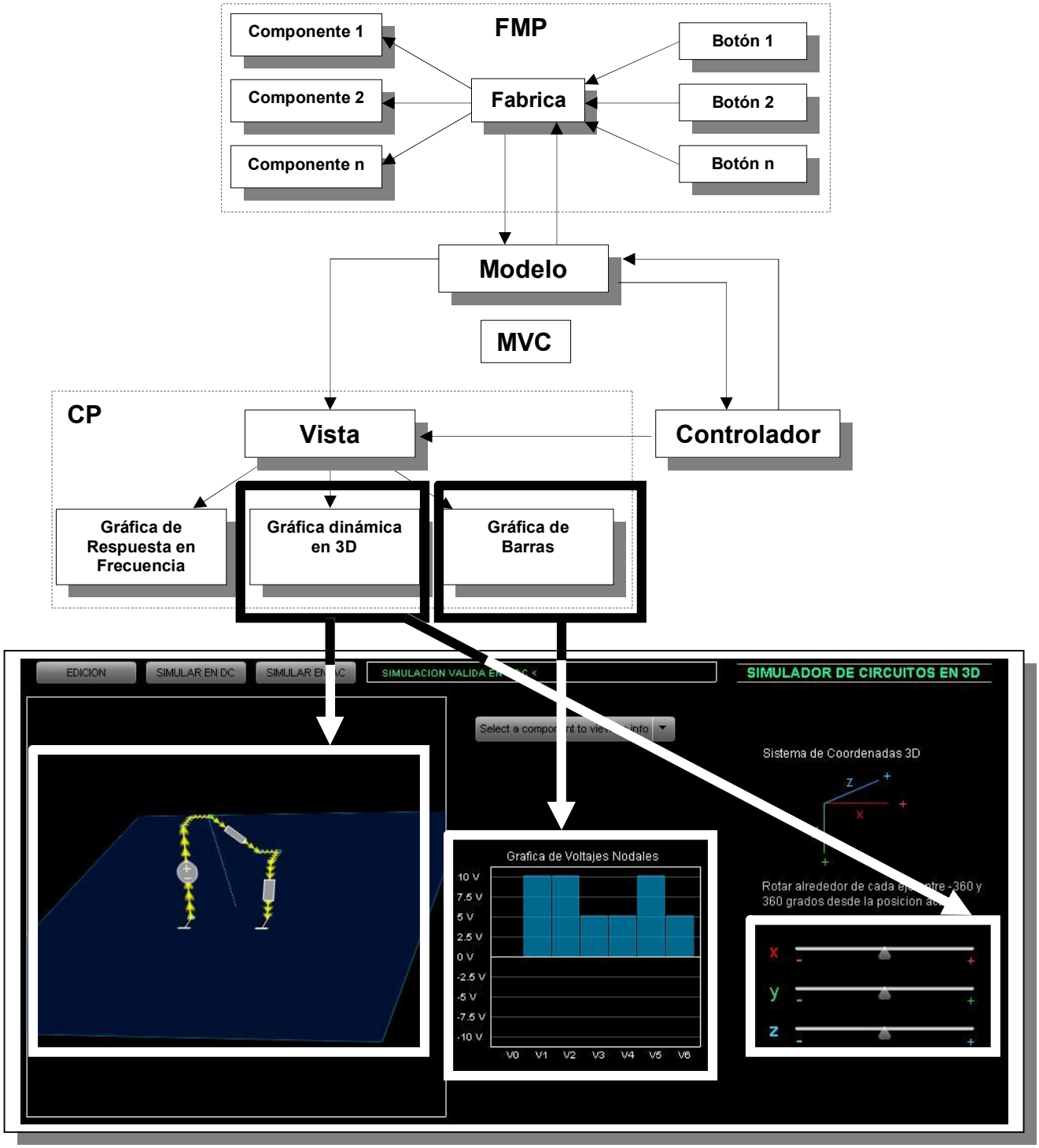


Figura III.6. Relación del patrón de programación **CP** con la pantalla del modo de **SIMULACIÓN EN DC**

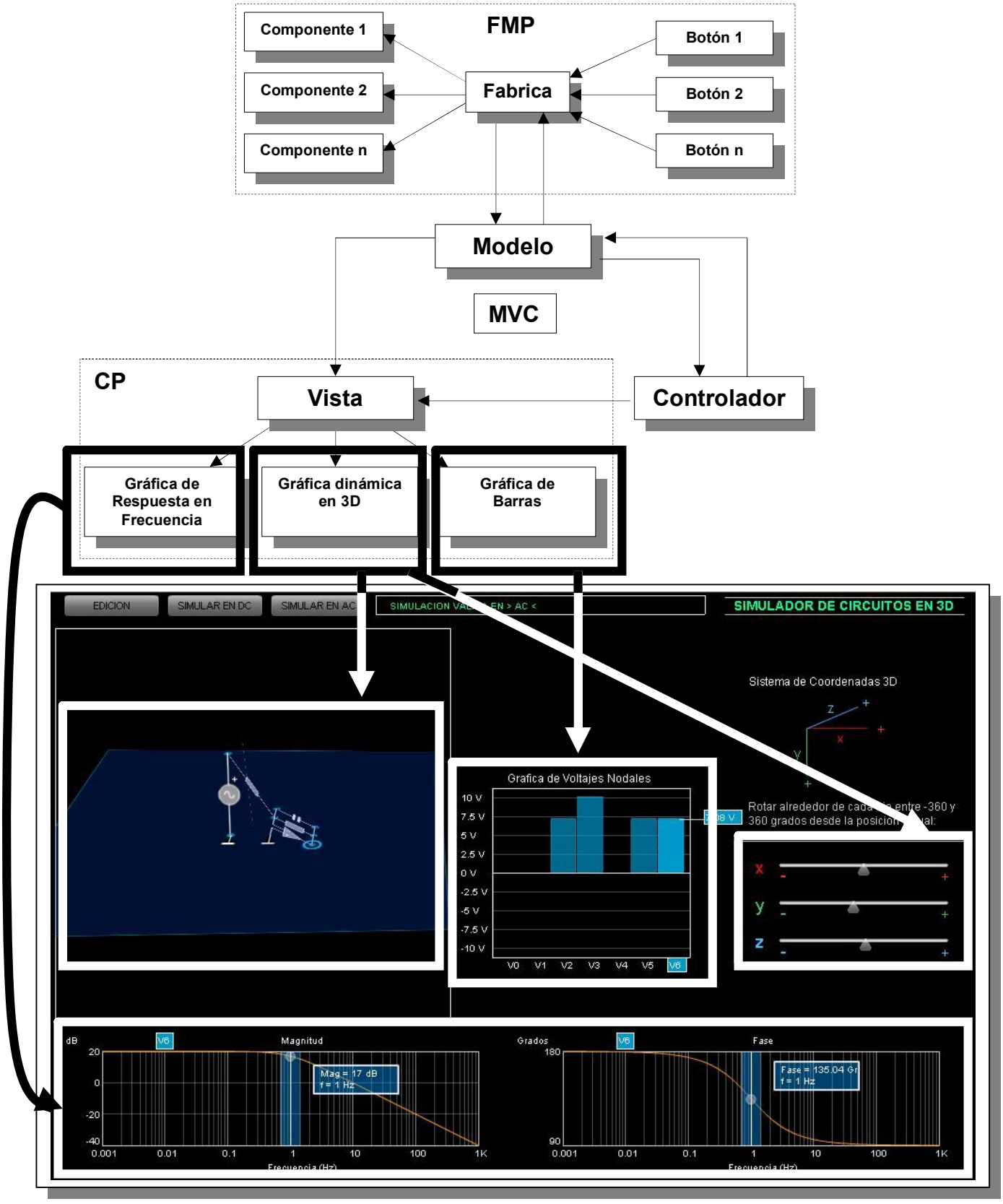


Figura III.7. Relación del patrón de programación **CP** con la pantalla del modo de **SIMULACIÓN EN AC**

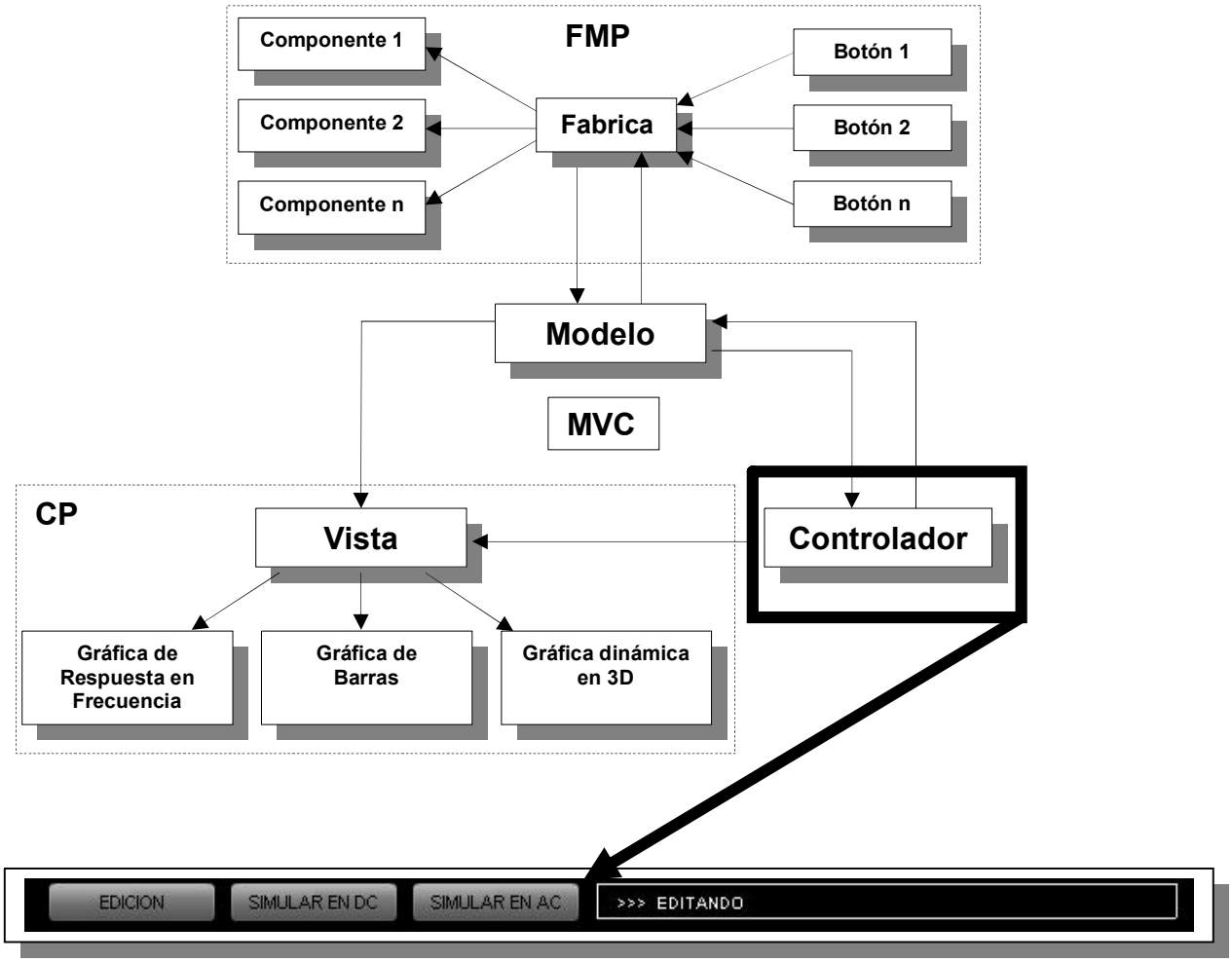


Figura III.8. Relación del módulo **Controlador** con la pantalla del simulador.

Los módulos **Fabrica**, **Modelo** y **Vista** son módulos que no tienen una representación gráfica en las pantallas del simulador. Estos módulos se encargan de diferentes funciones que controlan todo el proyecto.

En el siguiente capítulo se explica el funcionamiento global de todo el proyecto, así como el de cada una de las estructuras presentadas.

IV. FUNCIONAMIENTO

El funcionamiento del proyecto se puede representar como se muestra en el diagrama a bloques de la Figura IV.1 para la simulación en DC y en el diagrama a bloques de la Figura IV.2 para la simulación en AC.

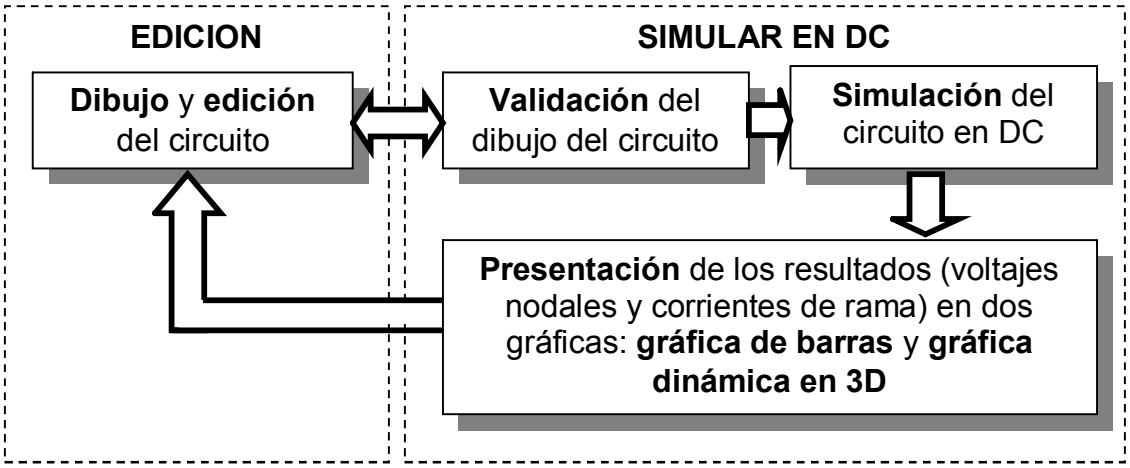


Figura IV.1. Diagrama a bloques del funcionamiento del proyecto para la simulación de un circuito en DC.

º

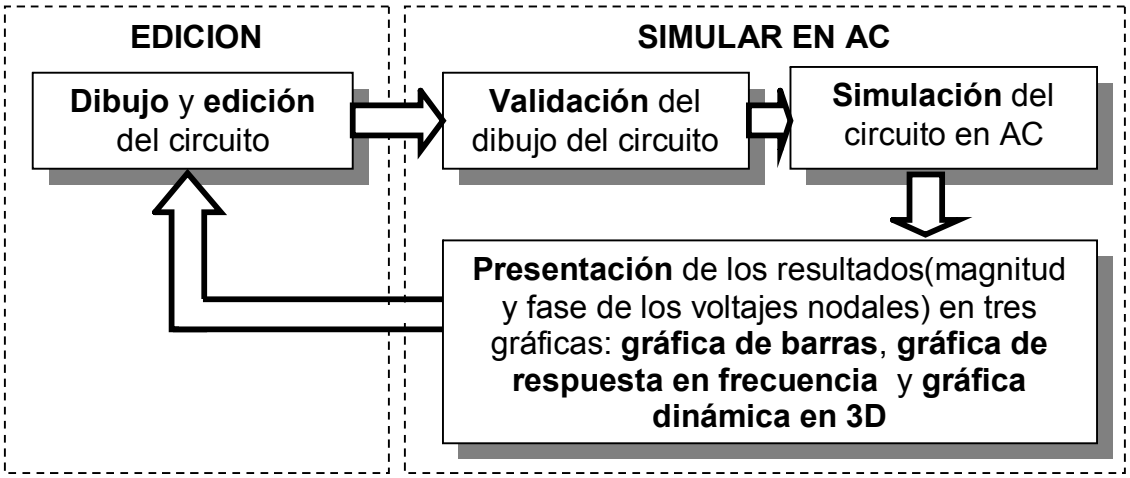


Figura IV.2. Diagrama a bloques del funcionamiento del proyecto para la simulación de un circuito en AC.

En ambas figuras se observa que el funcionamiento del proyecto es básicamente el mismo, las únicas diferencias son: el tipo de simulación y la presentación de los resultados en forma gráfica.

En cada gráfica el flujo de información siempre inicia en el modo de **EDICIÓN**, que es en donde el usuario **dibuja y edita el circuito** que desea simular. Posteriormente, en el bloque de **validación del dibujo del circuito**, se

comprueba que dicho dibujo cumpla con ciertos requisitos para poder simularse, ya sea en DC o en AC. Una vez validado dicho circuito, el bloque **simulación** se encarga de aplicar los algoritmos correspondientes para realizar la simulación deseada y, finalmente, el bloque de **presentación de resultados**, muestra los resultados de la simulación en diferentes formatos gráficos, siendo las **gráficas dinámicas en 3D**, las mas novedosas en este proyecto.

Se hace notar que, después de haber realizado alguna de las simulaciones disponibles, se puede regresar al estado de **EDICIÓN**, ya sea para modificar el circuito simulado, o dibujar un nuevo circuito, y comenzar nuevamente el ciclo.

El flujo de información mencionado anteriormente, se realiza en realidad entre las estructuras básicas del proyecto presentadas en el capítulo anterior. Considerando las tres estructuras principales **MVC**, **FMP** y **CP**, la generación e intercambio de información entre estas estructuras se realiza como se muestra en la Figura IV.3.

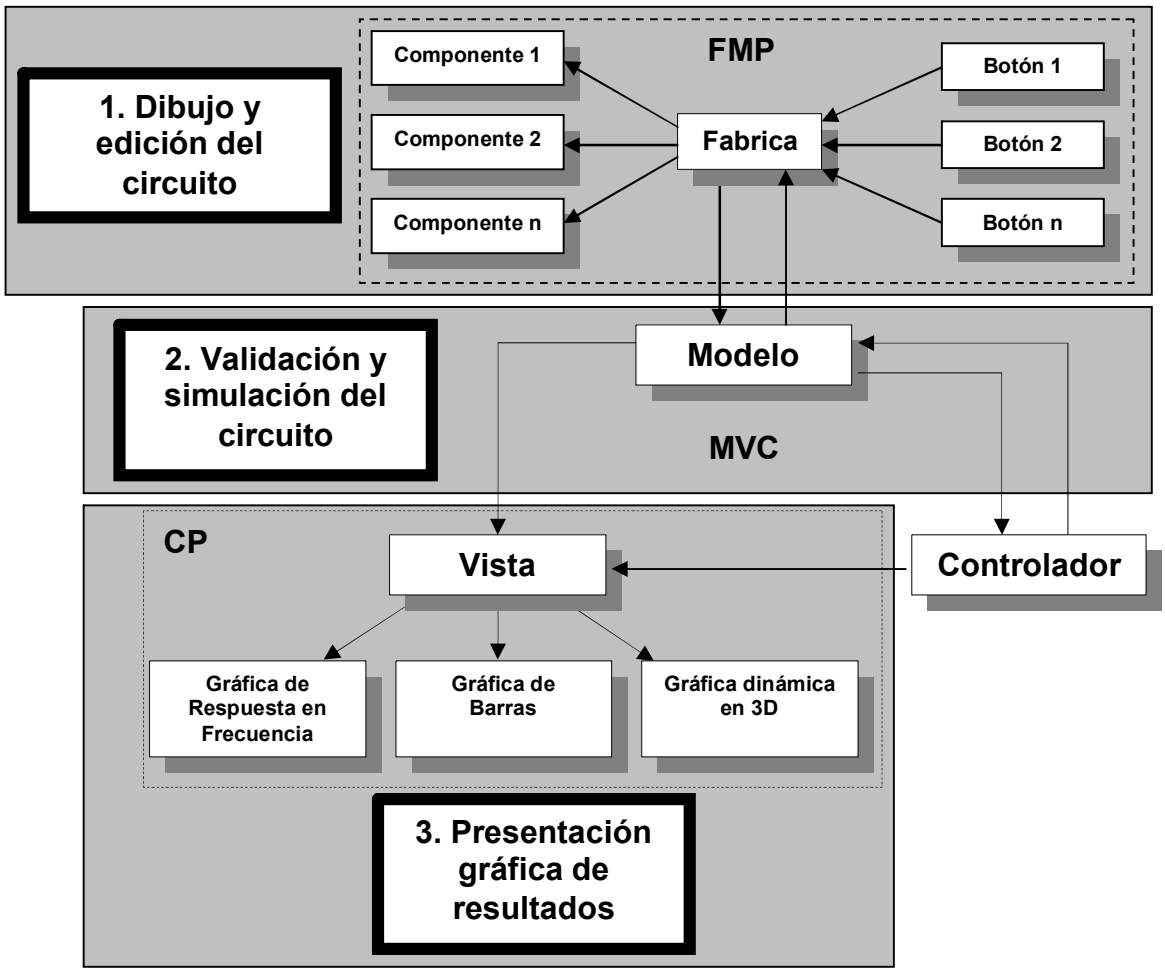
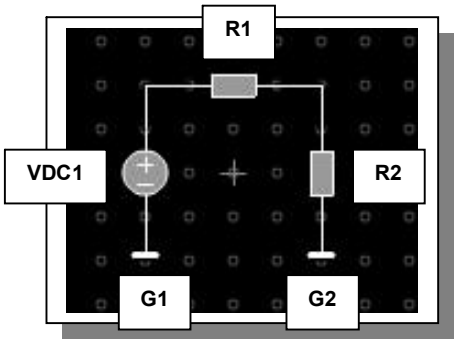


Figura IV.3. Generación e intercambio de información entre las estructuras **MVC**, **FMP** y **CP**.

IV.1 Dibujo y edición del circuito

Cuando el usuario dibuja un circuito en el Area de Dibujo, la estructura **FMP** se encarga de generar tantos **objetos** como **componentes** tenga el dibujo. Cada **objeto componente** guarda en si mismo la información sobre sus coordenadas de localización en el Area de Dibujo, así como sus parámetros asociados al componente en particular (identificador único, valor, tipo, etc.). También cada **objeto componente**, al aparecer dibujado en el Area de Dibujo, forma parte, automáticamente, de un **arreglo de objetos gráficos**, el cual podrá ser consultado por las estructuras **MVC** y **CP** cuando así lo requieran.

En la Figura IV.4 se muestra gráficamente, como ejemplo, el **dibujo de un circuito** y el **arreglo de objetos gráficos** generado para dicho circuito de ejemplo, mientras que en la Figura IV.5 se presenta el diagrama a bloques de la estructura del proyecto, indicando la ubicación del **arreglo de objetos gráficos** generado por la estructura **FMP**.



Arreglo de objetos gráficos

OBJETO**: FUENTE DE VOLTAJE DC**

- Identificador único: **VDC1**
- Información de coordenadas: x,y,z
- Valor de la fuente: 10 volts
- Tipo de Fuente: DC

OBJETO**: RESISTENCIA**

- Identificador único: **R1**
- Información de coordenadas: x,y,z
- Valor de la resistencia: 1 Kohm

OBJETO**: RESISTENCIA**

- Identificador único: **R2**
- Información de coordenadas: x,y,z
- Valor de la resistencia: 1 Kohm

OBJETO**: TIERRA**

- Identificador único: **G1**
- Información de coordenadas: x,y,z

OBJETO**: TIERRA**

- Identificador único: **G2**
- Información de coordenadas: x,y,z

Figura IV.4. **Dibujo de un circuito** y el **arreglo de objetos gráficos** correspondiente generado por la estructura **FMP**.

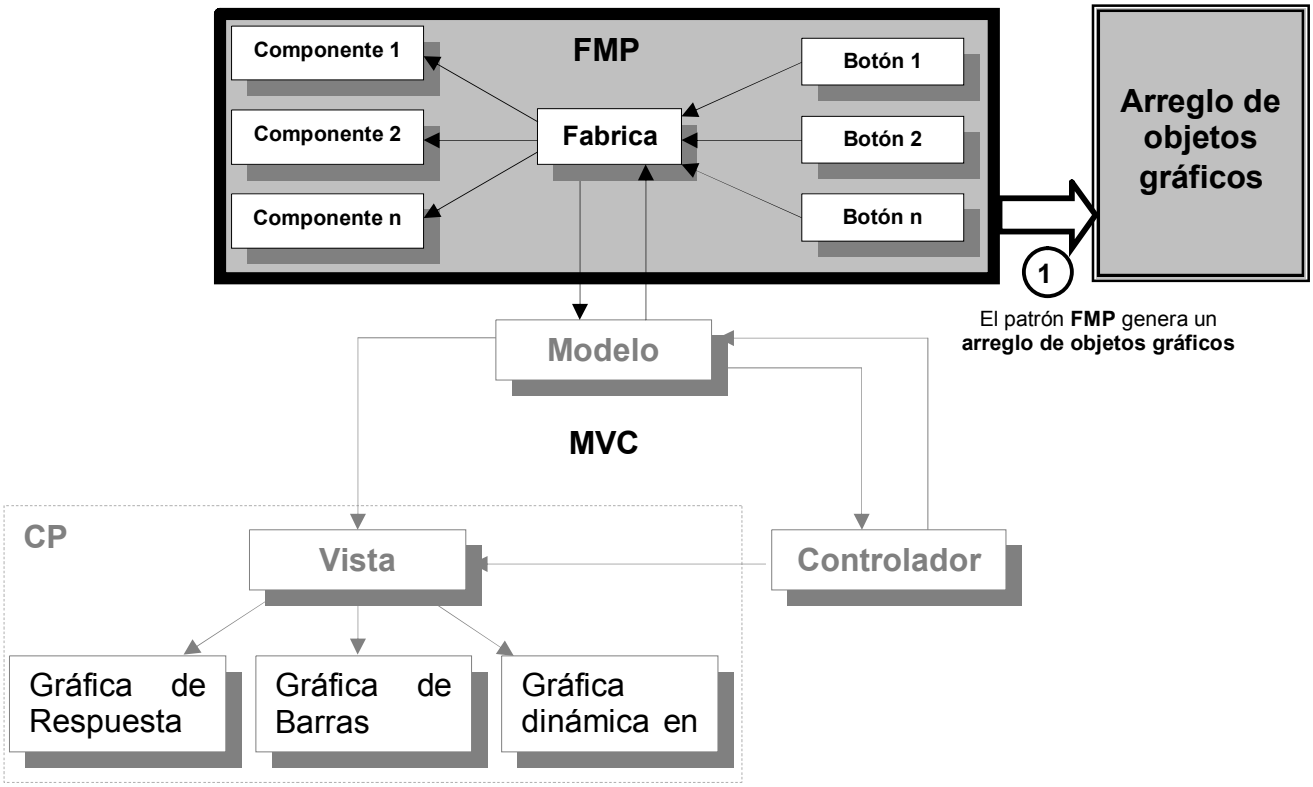


Figura IV.5. Ubicación del el **arreglo de objetos gráficos** generado por la estructura **FMP**.

IV.2 Validación del dibujo del circuito

Al momento de presionar el botón **SIMULAR EN DC** ó **SIMULAR EN AC**, el **modelo** lee el **arreglo de objetos gráficos** y lo **analiza** para buscar algún error en el circuito antes de simularlo. En la Figura IV.6 se muestra gráficamente este proceso. Para este análisis, el **Modelo** se apoya en una clase llamada **Manejador de red**, la cual realiza el análisis detallado de la red del circuito, reportando cualquier error al modelo, quien, a su vez, escribe un mensaje en la barra de estado, informando al usuario de la existencia de un error y su posible causa

Los mensajes de error que pueden aparecer en la barra de estado son los que se indican a continuación:

- "**NO EXISTE CIRCUITO QUE SIMULAR**" – Se presionó el botón "SIMULAR EN DC" ó "SIMULAR EN AC" sin tener dibujado algún circuito en el área de dibujo.
- "**CERRAR VENTANAS DE EDICION DE CADA COMPONENTE**" – Se encuentra abierta una o mas ventanas de edición de algun(os) componente(s).

- "**NO SE HA DEFINIDO EL NODO DE TIERRA**" – No está dibujado el símbolo de "tierra" en el área de dibujo.
- "**CIRCUITO NO CONECTADO A TIERRA**" – El circuito dibujado no tiene algún nodo conectado a "tierra".
- "**ELEMENTOS DESCONECTADOS DEL CIRCUITO**" – Existe uno o mas elementos que no están conectados entre sí.
- "**FUENTES EN CONEXION NO VALIDA**" – Fuente de voltaje en "corto circuito" ó fuente de corriente en "circuito abierto".

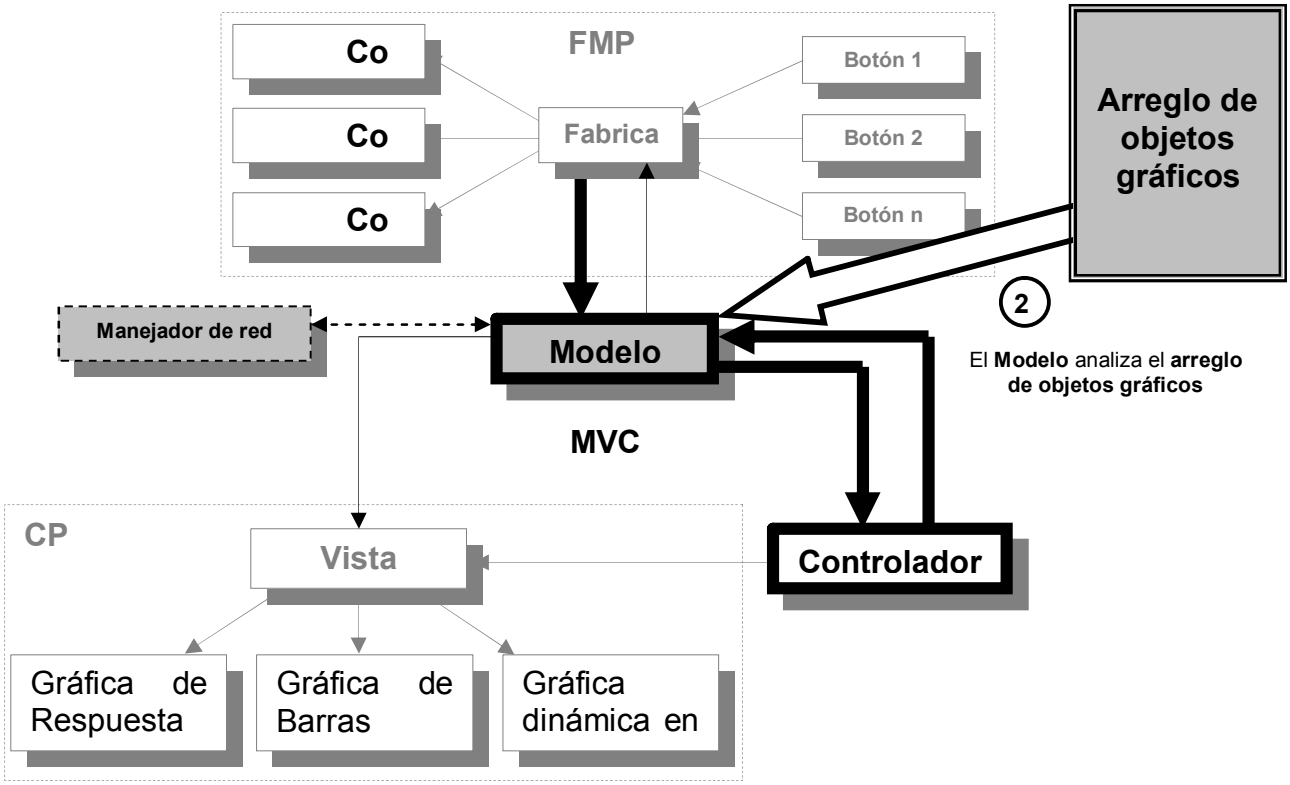


Figura IV.6. Análisis del **arreglo de objetos gráficos** por el bloque **Modelo**, auxiliado por el bloque **Manejador de red**

Si no hay errores o ya se corrigieron, la clase auxiliar, **Manejador de red**, genera un **archivo descriptor del circuito** con un formato similar al de SPICE, como el mostrado en la Figura IV.7:

```
C[0]   id: V1 c: 1,0  valor: 10      tipo: "DC"
C[1]   id: R1 c: 1,2  valor: 1000
C[2]   id: R2 c: 2,0  valor: 1000
```

Figura IV.7. **Archivo descriptor del circuito** generado por el bloque **Manejador de red**

En la Figura IV.8 se presenta la ubicación de este archivo en el diagrama de la estructura del simulador.

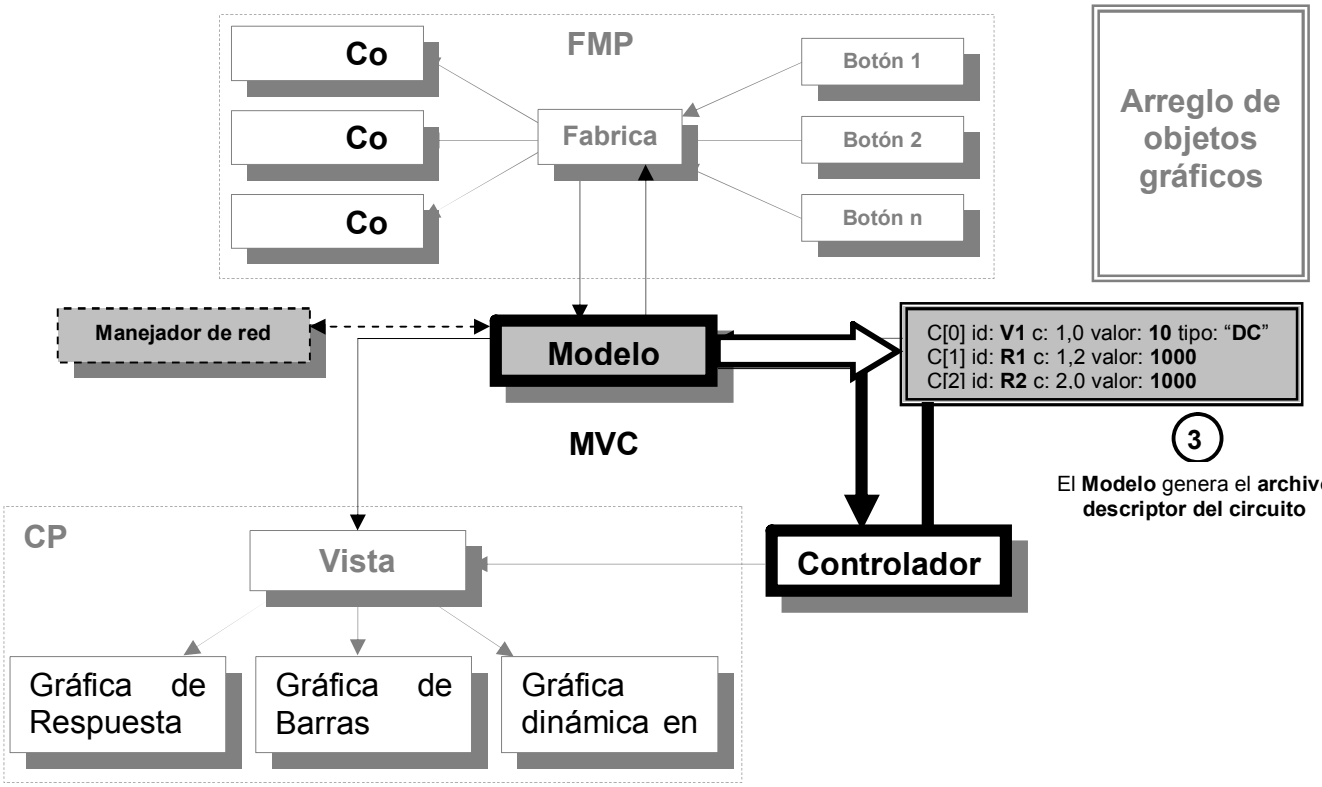


Figura IV.8. **Archivo descriptor del circuito** generado por el bloque **Modelo**, auxiliado por el bloque **Manejador de red**

Paralelamente con la generación del **archivo descriptor del circuito** el **Modelo** autoriza al **Controlador** para poder realizar el cambio de estado a **SIMULAR EN DC** ó **SIMULAR EN AC**, según lo haya solicitado el usuario.

IV.3 Simulación del circuito

Una vez generado el **archivo descriptor del circuito**, el **Modelo** lo envía a la clases auxiliares **Simulador en DC** o **Simulador en AC**, según corresponda, para realizar la simulación respectiva, como se observa en la Figura IV.9.

La simulación se realiza a través de un algoritmo llamado **Análsis Nodal Modificado** (**MNA**), el cual es un método de análisis de circuitos eléctricos adaptado para su procesamiento en una computadora. También se hace uso del algoritmo de **Gauss Jordan** para resolver el sistema de ecuaciones resultante.

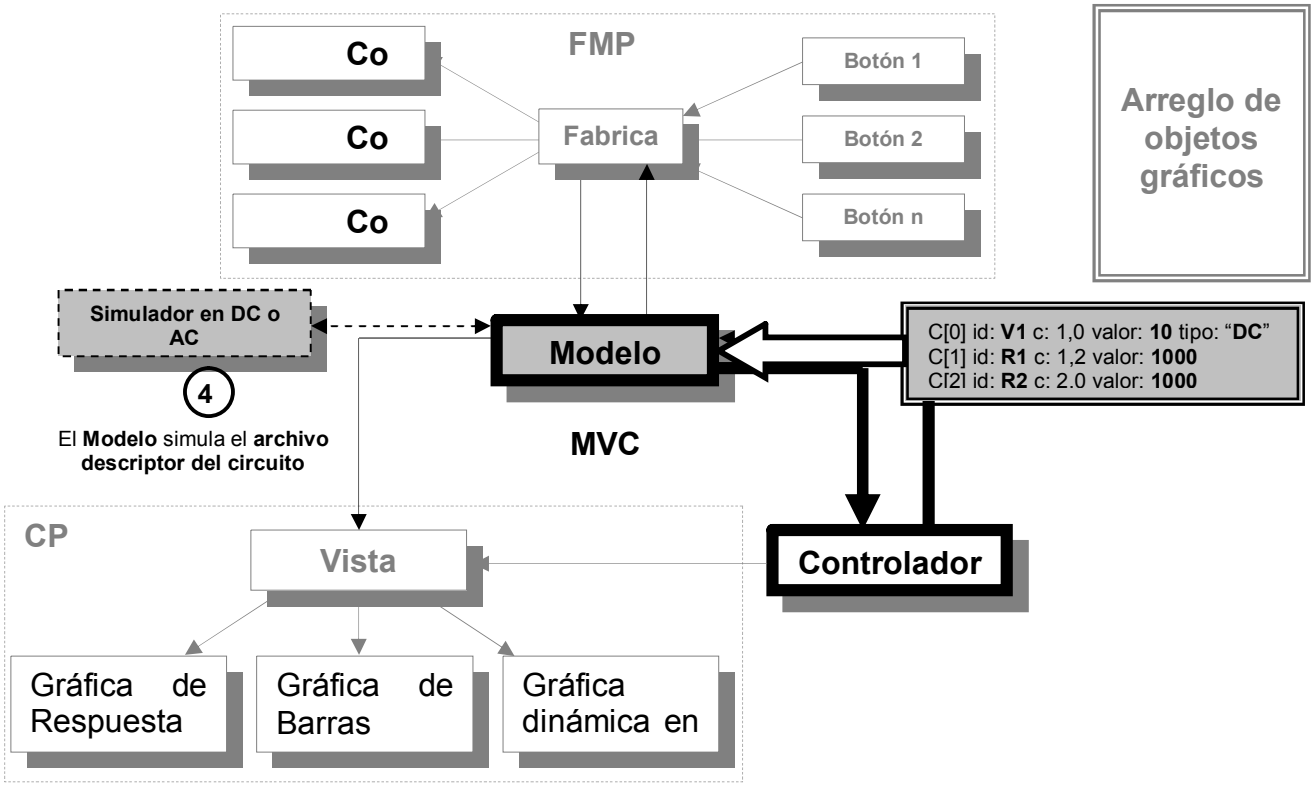


Figura IV.9. Simulación del **archivo descriptor del circuito** por el bloque **Modelo**, auxiliado por el bloque **Simulador en DC o AC**

Si existe algún error en la simulación, el Modelo lo reportará en la **barra de estado**. Los posibles errores de simulación pueden ser los siguientes:

- "**01:: ERROR EN LA SIMULACION: REVISAR EL CIRCUITO**" – Un voltaje nodal no pudo ser calculado.
- "**02:: ERROR EN LA SIMULACION: REVISAR EL CIRCUITO**" – Una corriente de rama no pudo ser calculada.
- "**03:: ERROR EN LA SIMULACION: REVISAR EL CIRCUITO**" - Un valor de magnitud y fase de algún voltaje nodal no pudo ser calculado.

Si se presenta alguno de estos errores, el **Modelo** no autorizará al **Controlador** para realizar el cambio de pantalla a **Vista** para desplegar los resultados, ya que éstos no existen, por lo que el estado del proyecto permanecerá en **EDICIÓN**, para que se pueda modificar el circuito

Si la simulación resultó exitosa, el **Modelo** autorizará el cambio de pantalla a **Vista**, a través del **Controlador**.

IV.4 Presentación de resultados

La presentación de los resultados se realiza mediante la estructura **CP** cuando el **Modelo** envía los resultados de la simulación al bloque **Vista**. Este bloque **Vista** constituye el nodo de la estructura **CP**, al cual se conectan dos o tres hojas de acuerdo al tipo de simulación: **gráfica de barras** y **gráfica dinámica en 3D** para la **simulación en DC** y, **gráfica de barras**, **gráfica de respuesta en frecuencia** y **gráfica dinámica en 3D** para la simulación en **AC**.

IV.4.1 Simulación en DC

El bloque **Vista** genera dos gráficas para representar los resultados de esta simulación: una **gráfica de barras** y una **grafica dinámica en 3D**, como se muestra en la Figura IV.10.

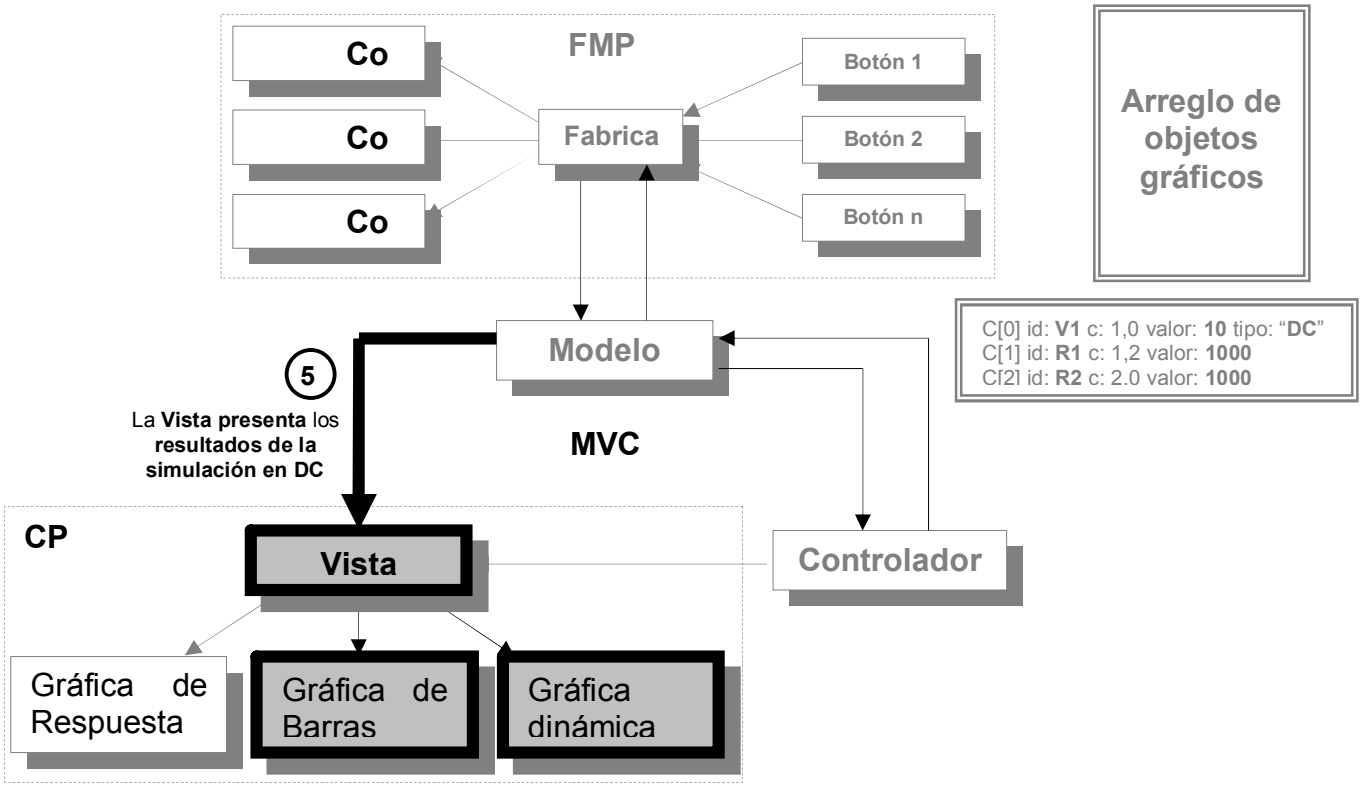


Figura IV.10. **Presentación de los resultados** de la **simulación en DC**

La **gráfica de barras** muestra los valores de los voltajes nodales en cada nodo del circuito. Es una gráfica interactiva. Al pasar el puntero del mouse sobre alguna barra suceden tres acciones: la barra se ilumina, el identificador del nodo correspondiente y el valor del voltaje nodal correspondiente aparecen resaltados en dicha gráfica y, finalmente, el nodo correspondiente a la barra seleccionada, aparece resaltado en la gráfica dinámica en 3D, mediante un círculo concéntrico alrededor de dicho nodo, para indicarle al usuario la ubicación de éste en el

circuito. Esta gráfica permite visualizar en conjunto, los voltajes nodales de todo el circuito.

La **gráfica dinámica en 3D** muestra, al mismo tiempo y en forma dinámica, la representación espacial de los voltajes nodales, y el flujo representativo de las corrientes de rama en todo el circuito. Los voltajes nodales se representan con la altura de dicho nodos con respecto al plano de referencia de potencial 0 volts, mientras que las corrientes de rama se representan como un flujo de flechas, cuyo "ancho" representa el valor comparativo de las corrientes en el contexto de todo el circuito. Como ya se mencionó en el párrafo anterior, al seleccionar una barra en la **gráfica de barras**, se genera un círculo concéntrico en el nodo respectivo de esta gráfica (la gráfica dinámica en 3D).

Ambas gráficas se muestran en la Figura IV.11. En esta figura de ejemplo, se puede observar la selección de una barra con el puntero del mouse y su efecto en ambas gráficas.

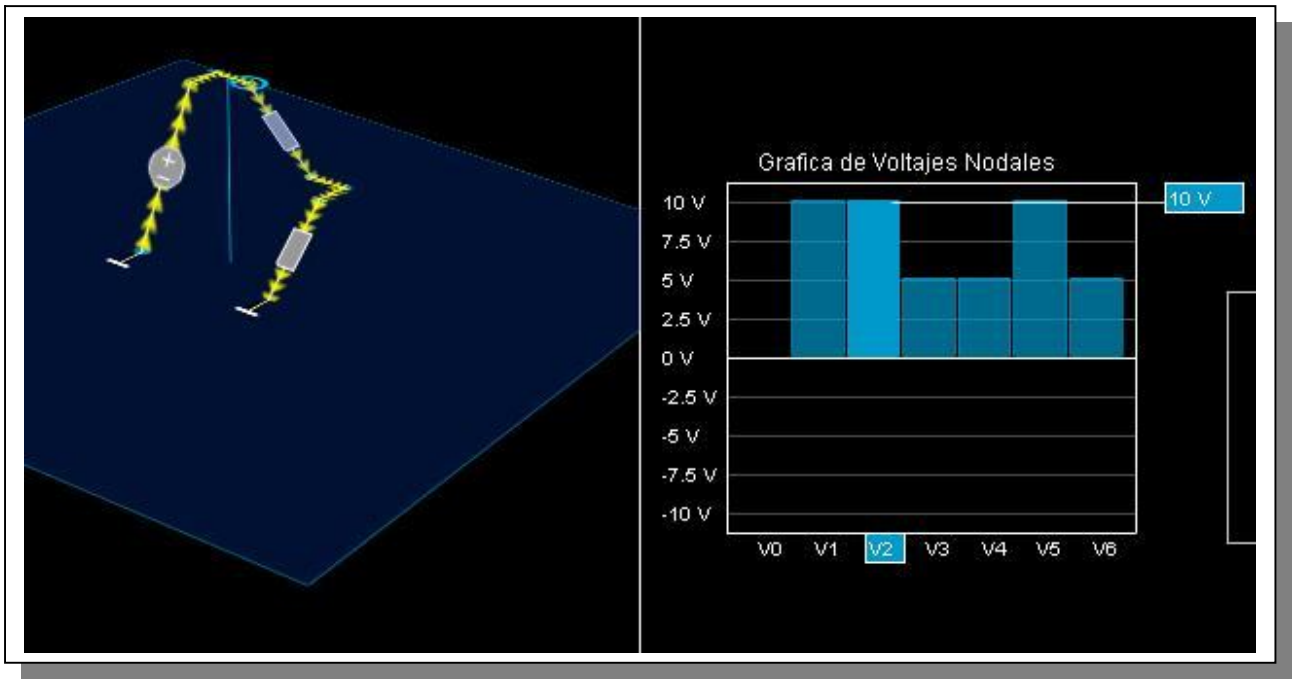


Figura IV.11. **Presentación de los resultados** de la **simulación en DC**, en una **gráfica de barras** y en una **gráfica dinámica en 3D**

Obsérvese que la gráfica dinámica en 3D contiene, además del plano de referencia de potencial 0 volts, una línea vertical perpendicular a dicho plano. Esta línea representa el sentido de valores positivos de los voltajes nodales correspondientes al circuito que se presenta.

IV.4.2 Simulación en AC

El bloque **Vista** genera tres gráficas para representar los resultados de esta simulación: una **gráfica de barras**, una **gráfica de respuesta en frecuencia** y una **grafica dinámica en 3D**, como se muestra en la Figura IV.12.

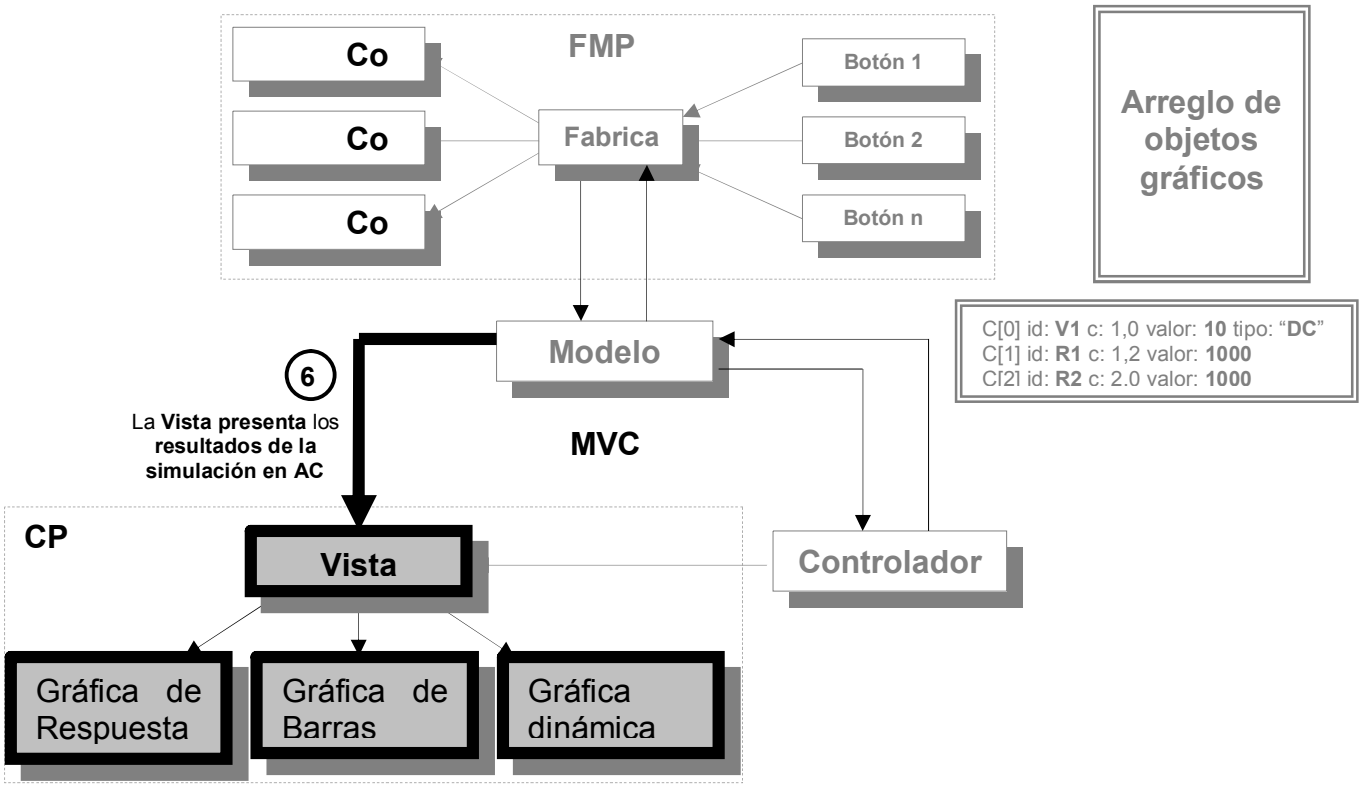


Figura IV.12. **Presentación de los resultados** de la **simulación en DC**

La **gráfica de barras** muestra los valores de las magnitudes de los voltajes nodales en cada nodo del circuito. Es una gráfica interactiva. Al pasar el puntero del mouse sobre alguna barra suceden cuatro acciones: la barra se ilumina, el identificador del nodo correspondiente y el valor de la magnitud del voltaje nodal correspondiente aparecen resaltados en dicha gráfica, se muestra la **gráfica de la respuesta en frecuencia** del nodo seleccionado y, finalmente, el nodo correspondiente a la barra seleccionada, se resalta en la **gráfica dinámica en 3D**, mediante un círculo concéntrico alrededor de dicho nodo.

La **grafica de respuesta en frecuencia** muestra la respuesta en frecuencia del nodo seleccionado en la **gráfica de barras**. Esta respuesta en frecuencia corresponde al diagrama de Bode, es decir, está formada por una gráfica de magnitud vs. frecuencia y otra gráfica de fase vs. frecuencia.

La **gráfica dinámica en 3D** muestra, al mismo tiempo y en forma dinámica, la representación espacial de los voltajes nodales complejos, mediante la representación de la magnitud a través de la altura de dicho nodos con respecto al

plano de referencia de potencial 0 volts, y la fase mediante su oscilación o movimiento cíclico en función del tiempo. Como ya se mencionó en el párrafo anterior, al seleccionar una barra en la **gráfica de barras**, se genera un círculo concéntrico en el nodo respectivo de esta gráfica (la gráfica dinámica en 3D).

Las tres gráficas se muestran en la Figura IV.13. En esta figura de ejemplo, se puede observar la selección de una barra con el puntero del mouse y su efecto en ambas gráficas.

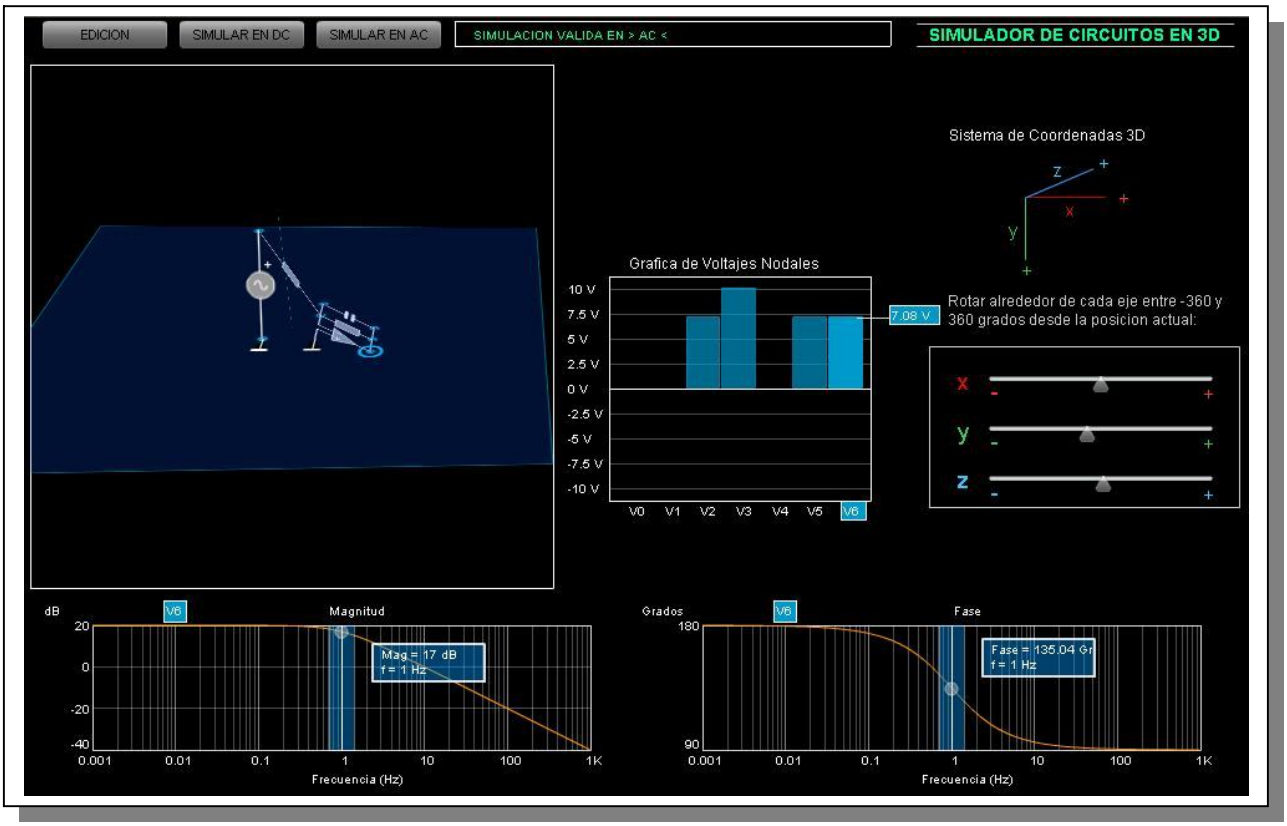
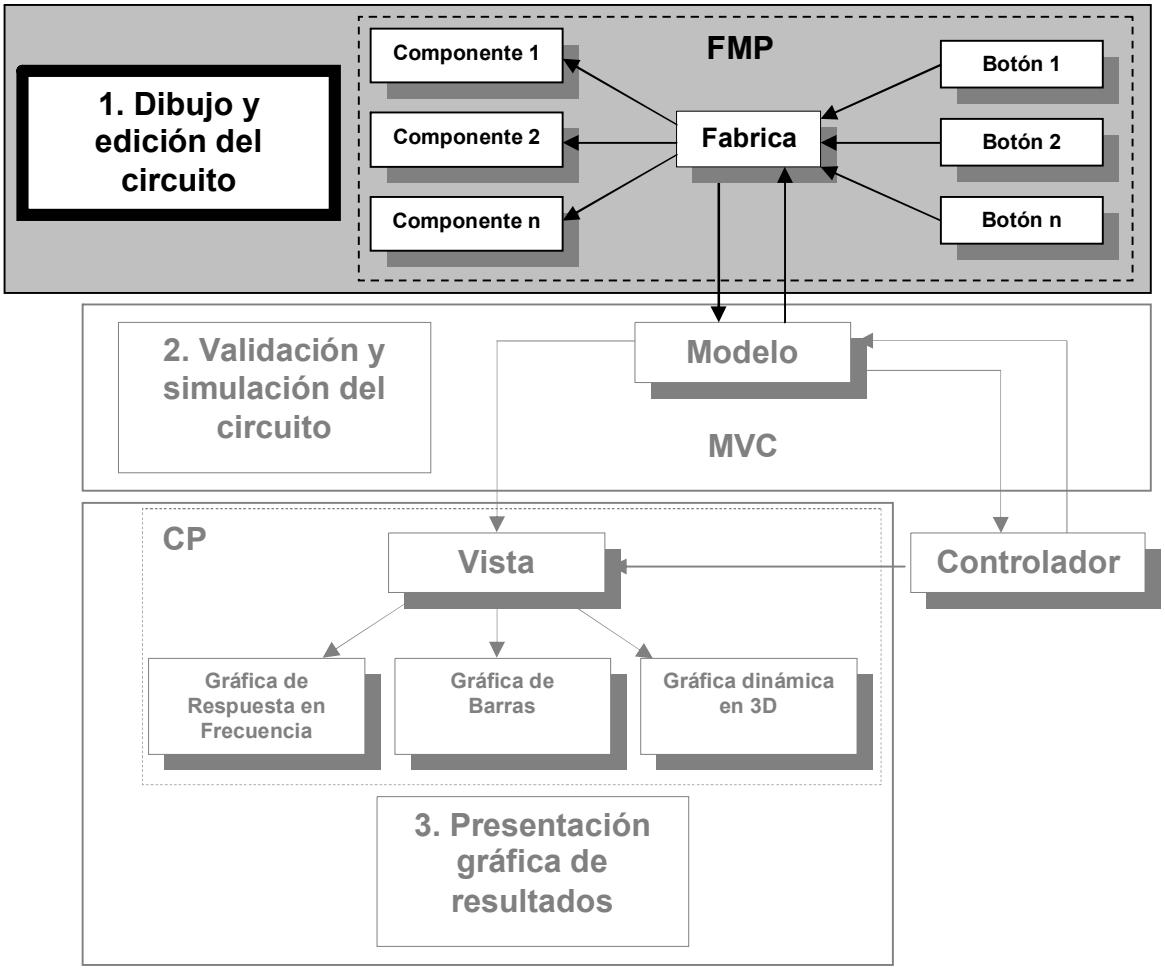


Figura IV.13. **Presentación de los resultados** de la **simulación en AC**: **gráfica de barras**, **gráfica de respuesta en frecuencia** y **gráfica dinámica en 3D**

Nuevamente se hace notar que la gráfica dinámica en 3D contiene, además del plano de referencia de potencial 0 volts, una línea vertical perpendicular a dicho plano. Esta línea representa el sentido de valores positivos de las magnitudes de los voltajes nodales correspondientes al circuito que se presenta.


1. Dibujo y edición del circuito
FMP
Componente 1
Componente 2
Componente n
Fabrica
Botón 1
Botón 2
Botón n
2. Validación y simulación del circuito
Modelo
MVC
CP
Vista
Controlador
Gráfica de Respuesta en Frecuencia
Gráfica de Barras
Gráfica dinámica en 3D
3. Presentación gráfica de resultados

V. Patrón de Programación METODO DE LA FABRICA (FMP).

El **FMP** se encuentra asociado al editor del circuito, como se ha explicado en los capítulos anteriores.

La función de esta estructura de programación es la de "fabricar" los componentes o circuitos solicitados por el usuario mediante el menú de componentes (fabricación de componentes) ó el menú de circuitos (fabricación de circuitos)

El proceso de fabricación se divide en tres etapas, como se muestra en la Figura V.1.

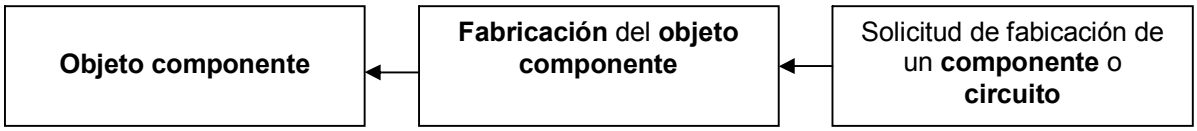


Figura V.1. Proceso de fabricación de un componente o circuito.

El proceso comienza con una solicitud, por parte del usuario, para fabricar algún componente o circuito. La fábrica se encarga entonces de fabricar dicho componente, generando como producto final el objeto componente solicitado

La fabricación de componentes (en la cual está basada también la fabricación de circuitos) se realiza, de manera mas detallada, como se muestra en la Figura V.2.

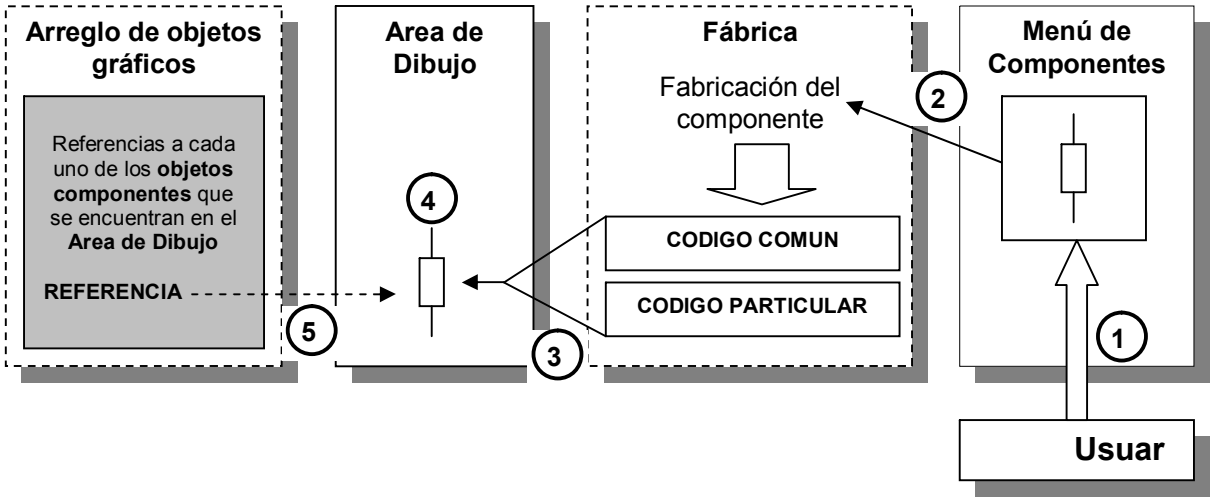


Figura V.2. Proceso detallado de fabricación de componentes.

En esta figura se lleva a cabo la siguiente secuencia de acciones:

1. El usuario solicita la fabricación de un componente presionando un botón del **Menú de Componentes**.
2. El botón presionado, a su vez, solicita a la **fábrica**, la producción de dicho componente.
3. La **fábrica** produce el **objeto componente** solicitado, creando un objeto cuyo código está formado por el **código común** que comparten todos los componentes, mas el **código particular** del componente solicitado.
4. El **símbolo gráfico** del componente se coloca en el **Area de Dibujo**, para que el usuario lo arrastre con el mouse y lo coloque en el lugar deseado para construir un determinado circuito. Se hace notar que, desde este momento, el **objeto componente** tiene todas las propiedades necesarias para comportarse como tal, es decir, si el símbolo es de una resistencia o un capacitor o una fuente, se comportará como tal al momento de simular el circuito
5. Las referencias a cada componente colocado en el **Area de Dibujo**, se guardan en el **arreglo de objetos gráficos**, para su uso posterior por los bloques **Modelo** y **Vista.**

V.1 Diagrama de clases

La definición e interconexión de clases en la estructura **FMP**, la cual permite generar la secuencia anterior, es muy interesante por el hecho de que dicha estructura **reutiliza el mismo código común** para todos los componentes que se manejan, además de permitir también, mediante la **extensión del código común**, proporcionar las características y funcionalidades particulares de cada componente. Así por ejemplo, una resistencia, un capacitor, una fuente de voltaje y una fuente de corriente, comparten un código común por ser componentes de dos terminales, sin embargo, cada uno de ellos tiene un comportamiento particular dependiendo de su función, lo cual hace necesario que cada componente extienda el código común mediante un código particular de éste.

Para lograr lo anterior, la estructura **FMP** cuenta con un mecanismo de selección del código correspondiente a cada componente en particular, pero si se quiere agregar al simulador un componente nuevo, la fábrica también tiene la flexibilidad necesaria para adaptarse a este cambio sin necesidad de **modifica**r el código ya existente en la fábrica, sino solamente **agregar** el nuevo código del nuevo componente.

Este mecanismo se puede visualizar mediante el diagrama de clases presentado en la Figura V.3, para la fabricación de una resistencia como ejemplo.

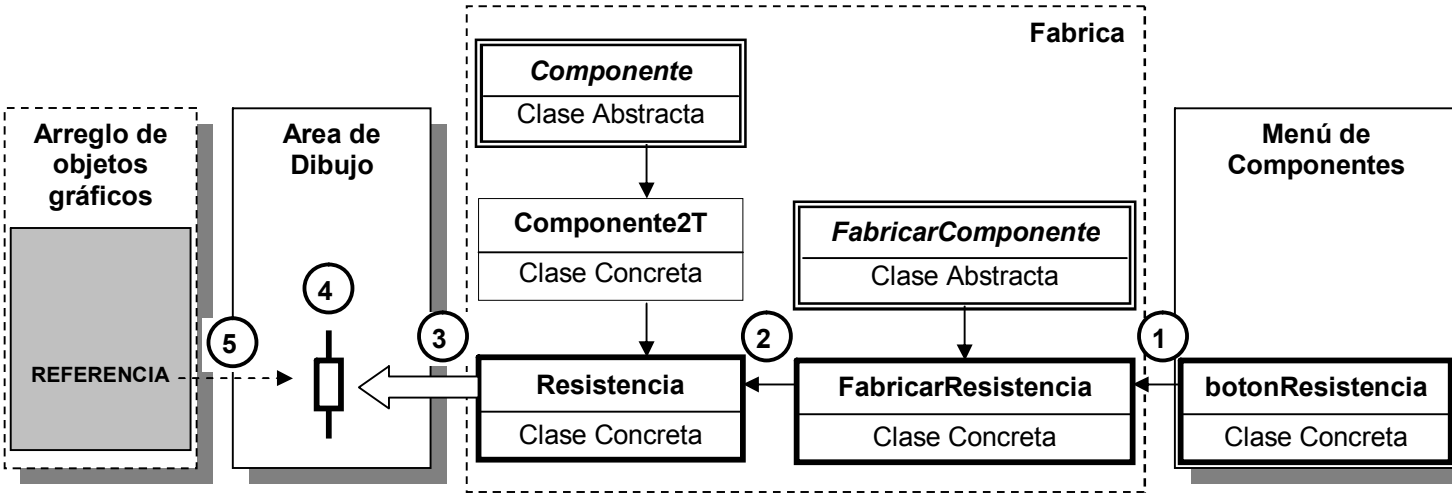


Figura V.3. Proceso de fabricación de una resistencia mediante la estructura **FMP**.

V.1.1 La Fábrica

En la Figura V.4 se muestra exclusivamente el diagrama de clases necesario para la fabricación de un objeto resistencia.

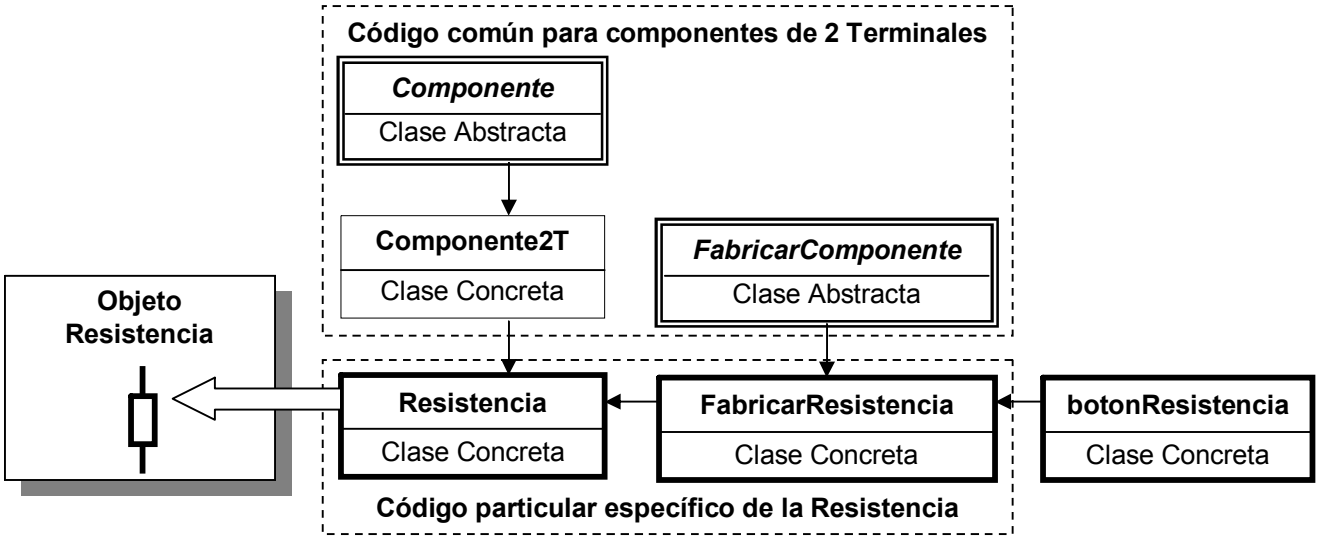


Figura V.4. **Diagrama de clases** de la estructura **FMP**, para la fabricación de un **objeto resistencia**.

En esta figura se representan las clases utilizadas mediante un rectángulo, cuyo **nombre de clase** se encuentra en la parte superior de éste, mientras que el **tipo de clase** se encuentra en la parte inferior del mismo.

La clase botonResistencia está asociada con el botón que aparece en el Menú de Componentes con el símbolo de una resistencia. Cuando el usuario presiona dicho botón, esta clase genera una petición de fabricación de una resistencia mediante el código que se muestra en la Figura V.5.

:: botonResistencia – Clase CONCRETA

```
public function respuestaBoton(event:MouseEvent):void {

        var componente:FabricarComponente = new FabricarResistencia();
        componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
}
```

Figura V.5. Código dela respuesta de la clase **botonResistencia** al ser presionado dicho botón por el usuario.

En la segunda línea de este código, se define una variable “componente” cuyo **tipo de variable** se asocia a la **clase abstracta FabricarComponente**. A la variable componente se le asigna el **objeto FabricarResistencia**, el cual se genera al “instanciar” la clase del mismo nombre. Lo importante aquí es el hecho de que la clase **FabricarResistencia** es una clase concreta que “extiende” a la **clase abstracta Fabricar Componente**, es decir, **FabricarComponente** es el **tipo de variable** y **Fabricar Resistencia** es el objeto asignado a dicha variable. En esta *definición* y *asignación* cruzada radica la clave de que esta estructura

FMP sea capaz de permitir la **adición** de cualquier otro componente nuevo **sin modificar su código de fabricación**, sino únicamente **agregando el nuevo código** de dicho componente. Esto es así, porque todos los componentes serán del mismo tipo (**FabricarComponente**), pero se les asociará un objeto diferente según el componente.

De la Figura V.4 se observa que, a su vez, la clase concreta **FabricarResistencia**, "instancia" a la clase concreta **Resistencia**, la cual constituye el conjunto de **propiedades gráficas** y de **propiedades eléctricas** propias de una resistencia física real. Este proceso se lleva a cabo mediante la tercera línea de código de la Figura V.5. En esta línea se hace un llamado al método local de la clase Fabricar Resistencia llamado Fabricar, que, como se observa en el código de la Figura V.6, es el encargado de **inicializar** a la clase concreta **Resistencia**.

En la Figura V.6 se presenta un resumen gráfico de la secuencia de fabricación de una resistencia. Las secuencias (**1**) y (**2**) se encargan de **construir la estructura** de la resistencia, mientras que las secuencias (**3**) y (**4**) se encargan de **inicializar** las **propiedades gráficas** y las **propiedades eléctricas** de la resistencia.

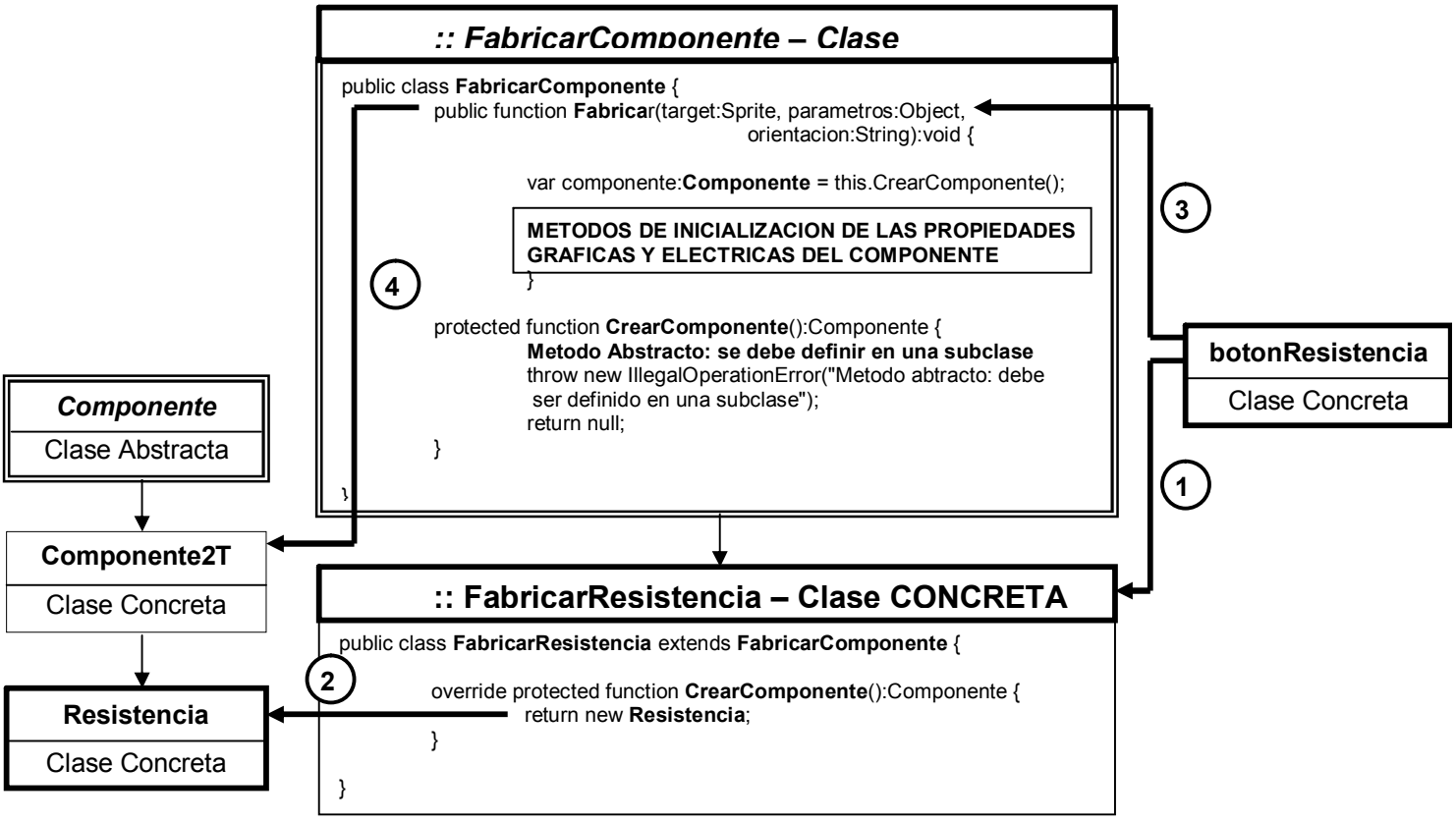


Figura V.6. Secuencia del proceso de fabricación de una resistencia, desde que se activa la clase **botonResistencia**, hasta que se **inicializa** el **objeto resistencia**.

V.1.2 Los botones del Menú de Componentes

Mediante esta misma técnica de producción de resistencias, es posible fabricar cualquiera de los componentes cuyos botones se encuentran en el Menú de Componentes. La extensión de esta técnica a otros componentes de dos terminales (2T), se muestra en la Figura V.7

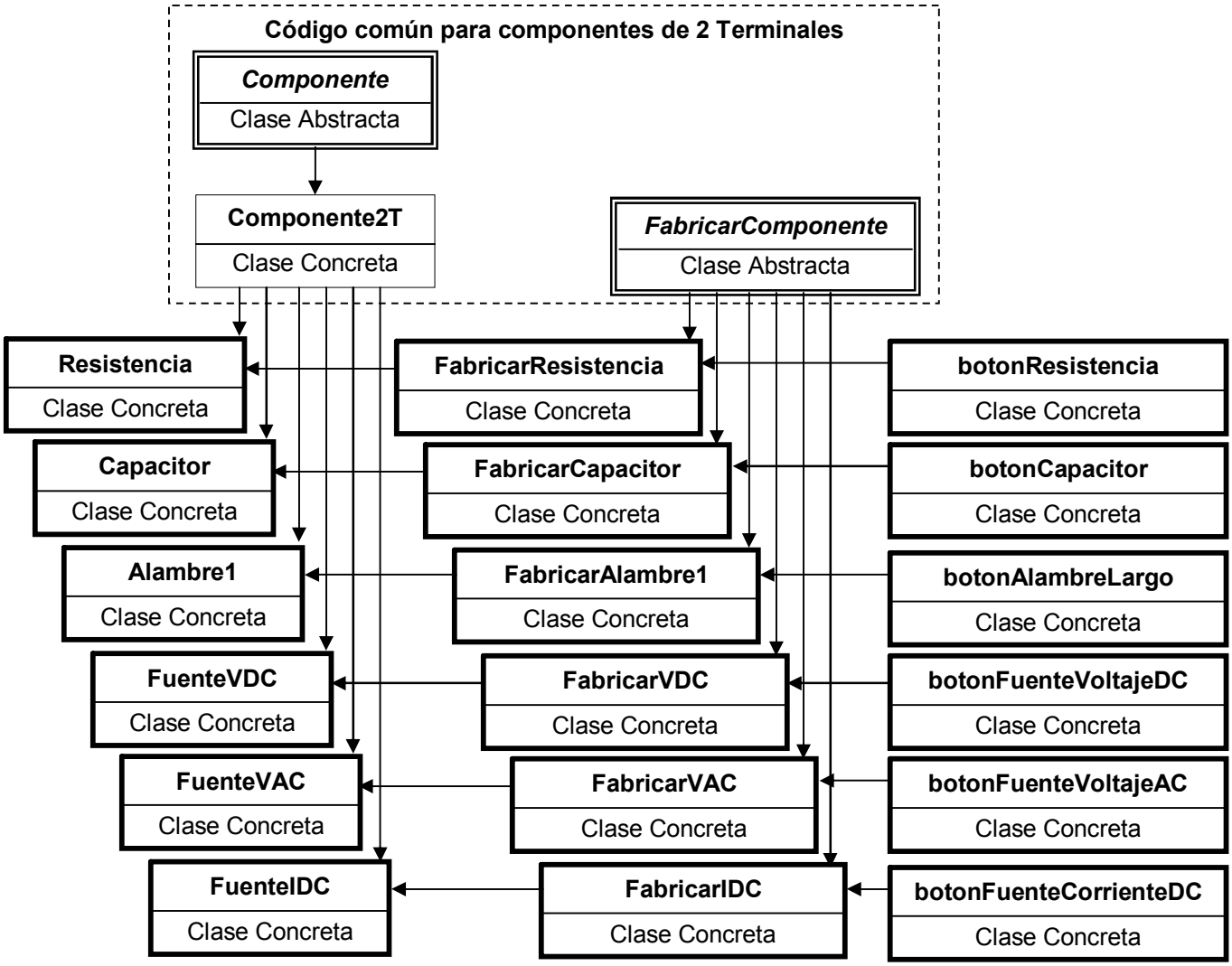


Figura V.7. **Diagrama de clases** de la estructura **FMP**, para la fabricación de diferentes **objetos componentes** de dos terminales (2T).

Para la fabricación de componentes de una terminal (tierra), alambre corto o componentes de tres terminales (transistores y amplificador operacional), se emplea un diagrama de clases similar al anterior, excepto que se emplean tres nuevas clase concretas: **Componente1T**, **AlambreCorto2T** y **Componente3**T respectivamente. En la Figura V.8 se muestra el diagrama de clases correspondiente.

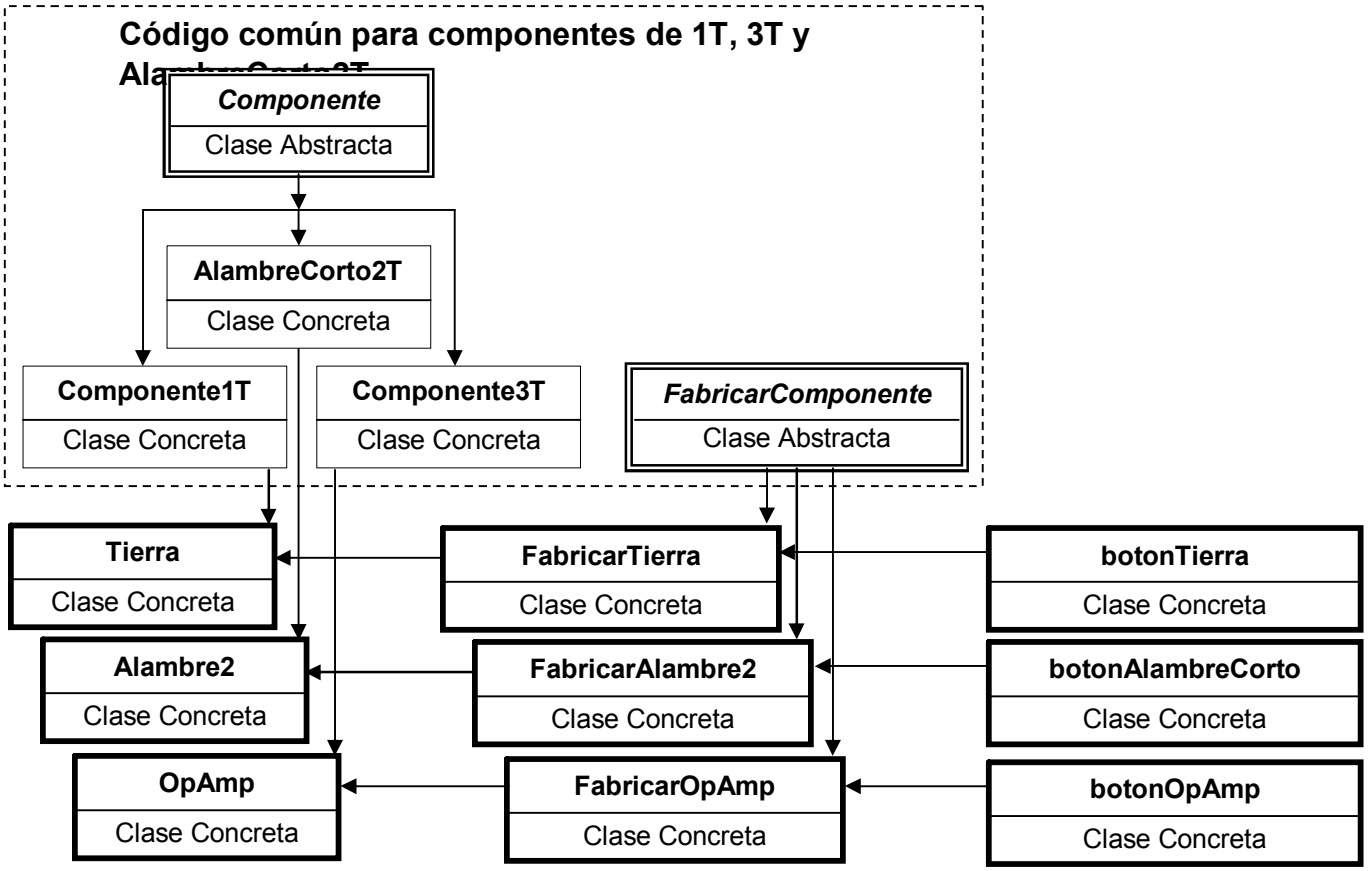


Figura V.8. **Diagrama de clases** de la estructura **FMP**, para la fabricación de diferentes **objetos componentes** de una terminal (**1T**), tres terminales (**3T**) y.**AlambreCorto**

Cada una de las clases que se encuentran en la columna de la derecha en las Figuras V.7 y V.8, al "instanciarse" generan los **objetos componentes** respectivos al componente de que se trate. Por ejemplo, **Capacitor**, **Tierra**, **OpAmp**, etc

V.1.3 Los botones del Menú de Circuitos

Los botones del Menú Circuitos se encargan, mediante un proceso similar al de la sección anterior, de "instanciar" cada uno de los objetos componentes definidos en una clase llamada **descriptor de circuitos**. Esta clase contiene la descripción de todos los componentes que constituyen un determinado circuito. Esta descripción consiste en la definición del tipo de componente, su posición física en el Area de Dibujo y los valores de todos los parámetros para cada componente. En la Figura V.9 se muestra una parte del código correspondiente al descriptor de un circuito divisor de voltaje resistivo, formado por dos resistencias, alambres y una fuente de voltaje de DC.

:: circuitoDivisor – Clase CONCRETA (fragmento)

```
public class circuitoDivisor1 {

	public function circuitoDivisor1():void {
		// --- Constructor ---
	}

	public function descriptorCircuito():Array {
		// ============================================================
		// Descriptor del circuito:
		//
		// ============================================================

		var circuito:Array = new Array();
		var descriptor:Object;

		descriptor = new Object();
		descriptor.datosComponente = new Array();

		descriptor.datosComponente[0] = new Array();
		descriptor.datosComponente[0][0] = "V0";
		descriptor.datosComponente[0][1] = 218;
		descriptor.datosComponente[0][2] = -40;
		descriptor.datosComponente[0][3] = 0;
		descriptor.datosComponente[0][4] = 0;
		descriptor.datosComponente[1] = [197,-40,-20,0,];
		descriptor.datosComponente[2] = [239,-40,20,0,];
		descriptor.i_tipoFuente = "DC";
		descriptor.u_valorComponente = 10;
		descriptor.orientacion = "N";

		circuito.push(descriptor);

		descriptor = new Object();
		descriptor.datosComponente = new Array();

		descriptor.datosComponente[0] = new Array();
		descriptor.datosComponente[0][0] = "W21";
		descriptor.datosComponente[0][1] = 197;
		descriptor.datosComponente[0][2] = -40;
		descriptor.datosComponente[0][3] = -20;
		descriptor.datosComponente[0][4] = 0;
		descriptor.datosComponente[1] = [176,-40,-40,0];
		descriptor.datosComponente[2] = [197,-40,-20,0];
		descriptor.tamanio = "corto";
		descriptor.u_valorComponente = 0;
		descriptor.orientacion = "N";

		circuito.push(descriptor);

		descriptor = new Object();
		descriptor.datosComponente = new Array();

		descriptor.datosComponente[0] = new Array();
		descriptor.datosComponente[0][0] = "R6";
		descriptor.datosComponente[0][1] = 222;
		descriptor.datosComponente[0][2] = 40;
		descriptor.datosComponente[0][3] = 0;
		descriptor.datosComponente[0][4] = 0;
		descriptor.datosComponente[1] = [201,40,-20,0,];
		descriptor.datosComponente[2] = [243,40,20,0,];
		descriptor.u_valorComponente = 1000;
		descriptor.orientacion = "N";

		circuito.push(descriptor);
```

Figura V.9. Fracción del **código del descriptor** de un circuito divisor de voltaje resistivo.

En la figura anterior, cada sección de código genera el componente respectivo hasta crear el circuito completo. Cada componente, a su vez, se genera mediante todo el procedimiento descrito en la sección anterior.

V.1.4 Los objetos componentes

Cada **objeto componente** que se genera a través de la fábrica, tienen su representación gráfica en forma de símbolo, el cual es el que aparece en el Area de Dibujo. Los símbolos disponibles son los que se muestran en la Figura V.10, y se encuentran en el Menú de Componentes.

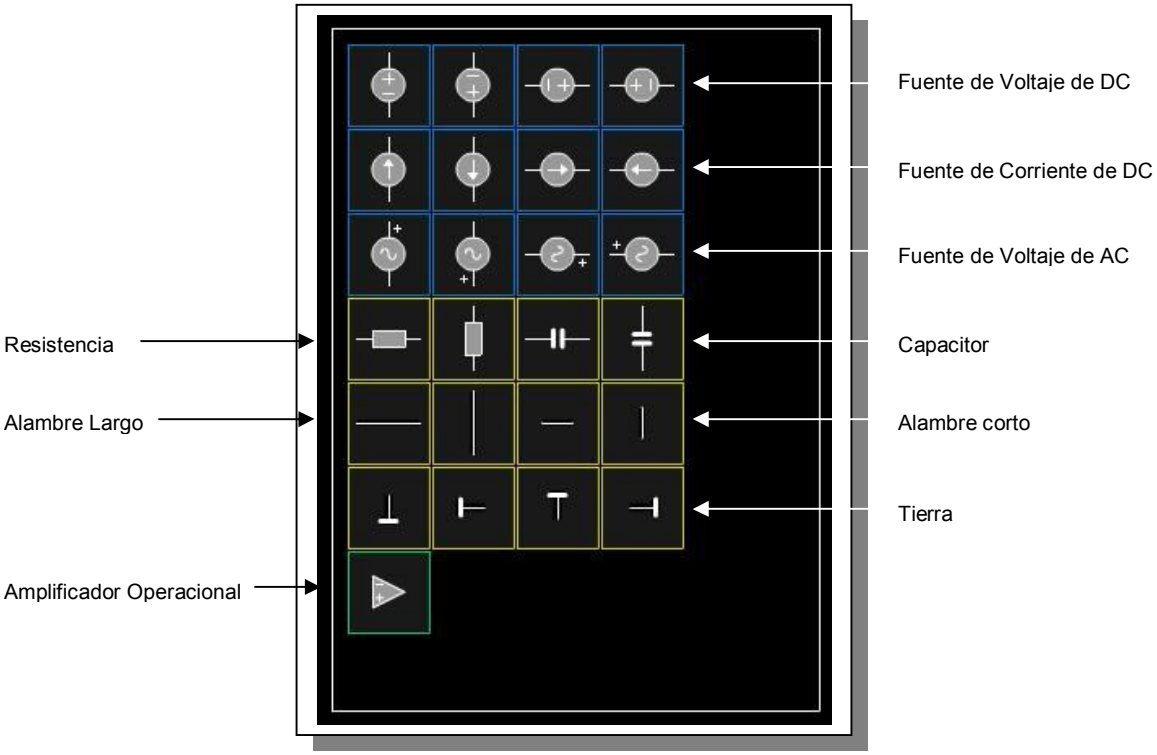


Figura V.10. Símbolo disponibles para componentes.

En la Figura V.11 se observa que todos los objetos componentes, sin importar su tipo, se originan de una **clase abstracta** llamada **Componente**. Esta **clase base** la **extienden** las cuatro **clases concretas, Componente1T**, **Componente2T** y **Componente3T**, para componentes de una, dos y tres terminales respectivamente, y la clase concreta **AlambreCorto2T** para alambres cortos de dos terminales. Solamente la **tierra** es un componente de una terminal y el **amplificador operacional** es un componente de tres terminales. Los demás componentes tienen dos terminales.

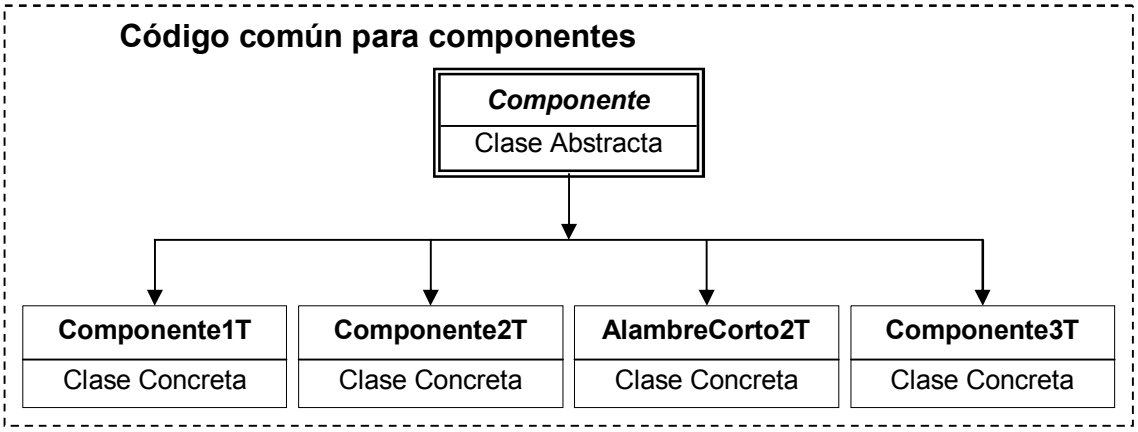


Figura V.11. Diagrama de **clases comunes** a todos los componentes disponibles en el **Menú de Componentes**.

Cada objeto componente en particular, **extiende** nuevamente a la clase que le corresponde de acuerdo a su número de terminales. Por ejemplo, la **tierra extiende** a la clase concreta **Componente1T**, el **capacitor extiende** a la clase conreta **Componente2T**, el **amplificador operacional extiende** a la clase concreta **Componente3T**. En la Figura V.12 se presenta un resumen gráfico de los **objetos componentes** disponibles y la **clase concreta** que **extienden**.

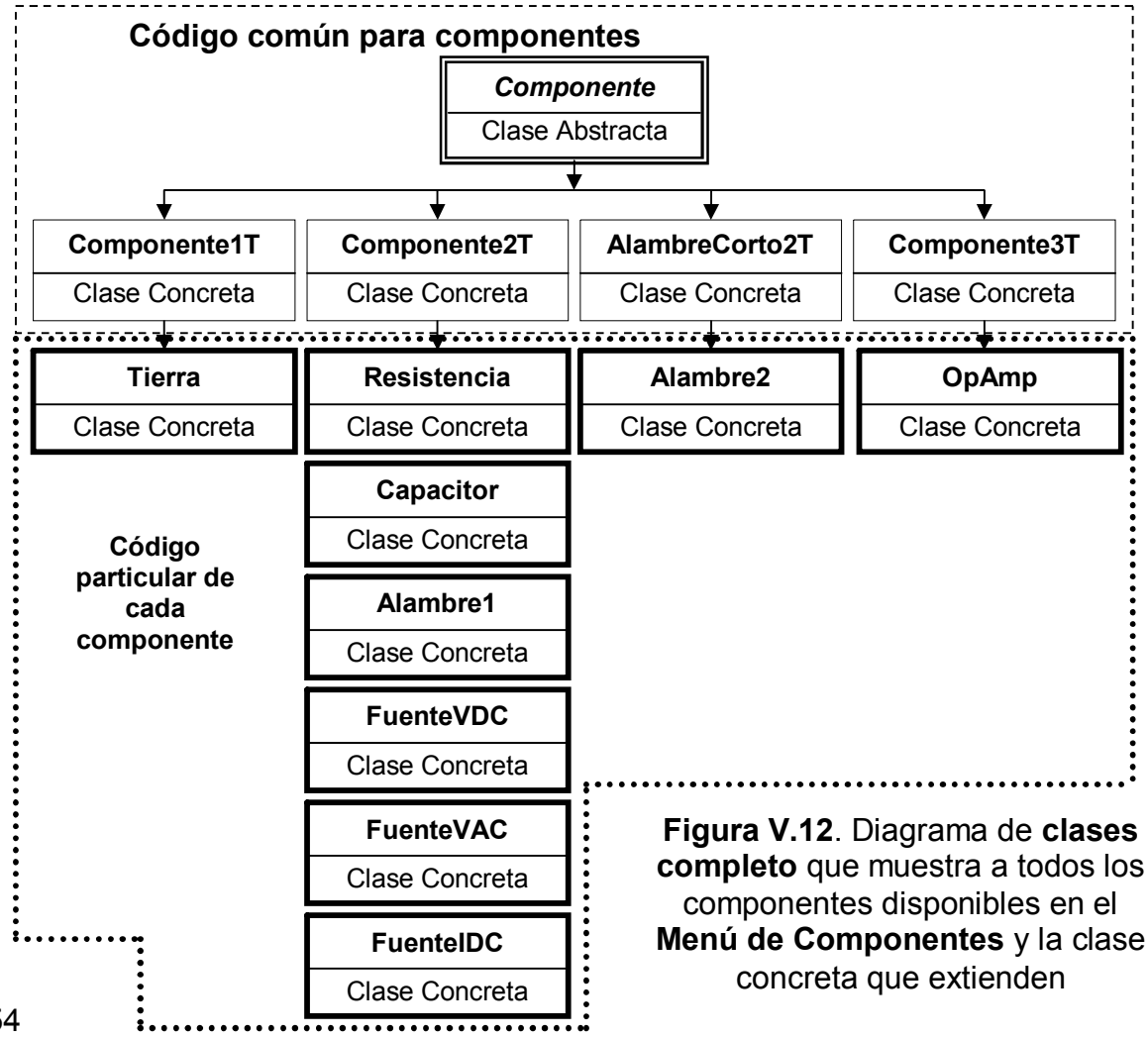


Figura V.12. Diagrama de **clases completo** que muestra a todos los componentes disponibles en el **Menú de Componentes** y la clase concreta que extienden

Cuando se dibuja un circuito en el **Area de Dibujo**, por el hecho de colocarlo dentro de esta área, a cada **objeto componente** que forme parte del circuito, se le asigna una **referencia** que queda guardada en un **arreglo de objetos gráfico**. Este arreglo es muy importante, pues constituye el punto de conexión con las estructuras **MVC** y **CP**, como se explicará mas adelante.

V.2 La clase abstracta Componente

La definición de **abstracta** asignada a esta clase, obliga al programador a **extenderla** y no **instanciarla**, además de que crea un **nodo de programación** que permite mucha **flexibilidad**, en el sentido de que de este nodo se pueden **colgar** diferentes componentes de una, dos o tres terminales, con diferentes comportamientos.

La clase abstracta **Componente** está estructurada con las **propiedades** y **métodos** que son comunes a todos los componentes sin importar su número de terminales. En la Figura V.13 se muestra la estructura del código de la esta clase abstracta.

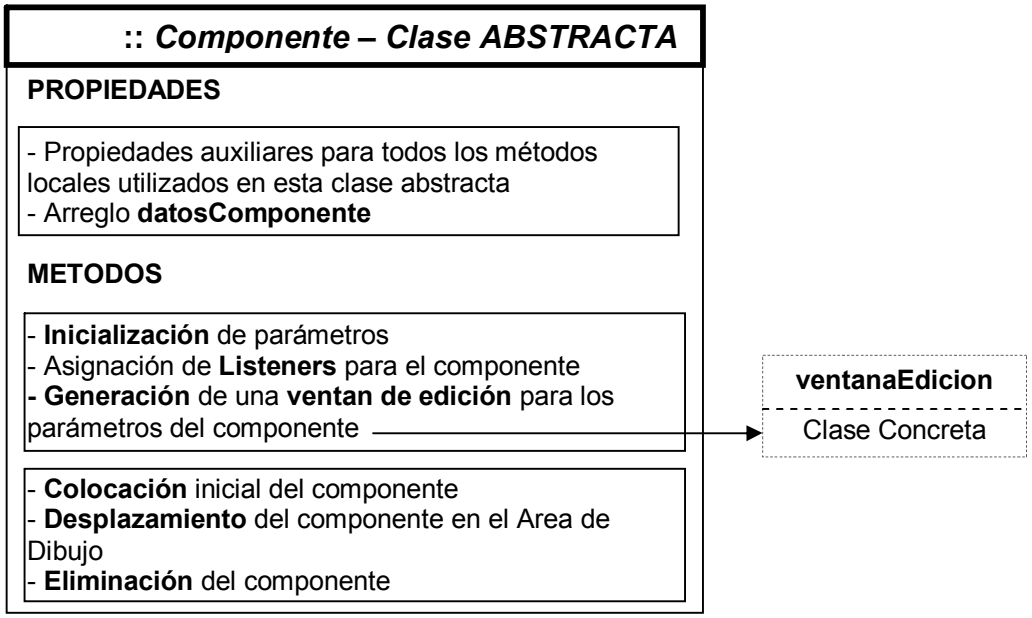


Figura V.13. Estructura del código de la clase abstracta **Componente**.

Los **métodos** que aparecen en el código de la figura, constituyen dos grupos de funciones comunes a todos los componentes, cuyas funciones son:

1. Asignar al componente un **identificador único**, recibir información sobre dimensiones y coordenadas de posición válidas del **Area de Dibujo**, inicializar **listeners** y crear una **ventana de edición** de parámetros propios del componente.

2. Definir la **colocación ini**cial del componente después de que el usuario presionó el botón correspondiente, permitir el **desplazamiento del componente** en el **Area de Dibujo** (bajo ciertas reglas de movimiento) y permitir la **eliminación del componente** por parte del usuario.

En esta clase se define un **propiedad** (tipo arreglo) llamada **datosComponente**, que guarda información de la localización física espacial (estática y dinámica) del componente y sus terminales. La estructura de este arreglo es la que se muestra en la Figura V.14.

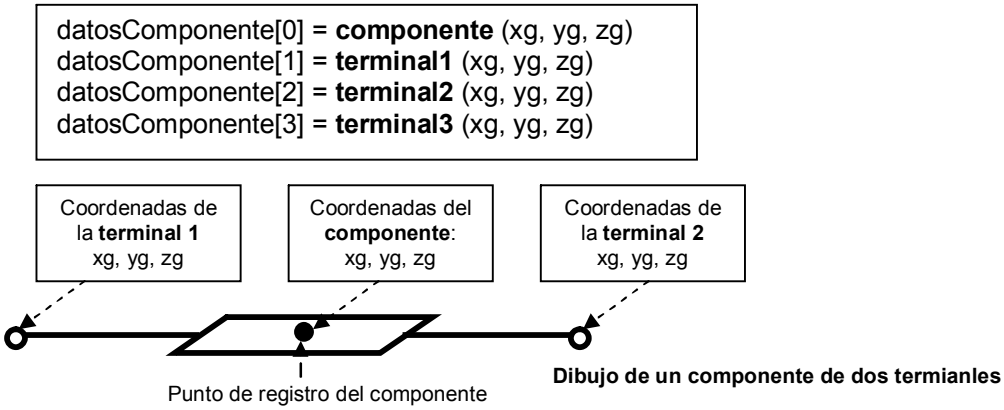


Figura V.14. Estructura de la **propiedad datosComponente** de la **clase abstracta Componente**.

La **clase abstracta Componente** en realidad sólo genera el primer renglón de la estructura anterior, es decir, el que corresponde a **datosComponente[0]**. Los siguientes renglones de esta estructura los generan cada una de las clases concretas que la extienden (**Componente1T, Componente2T, AlambreCorto2T** y **Componente3T**), como se explica en la siguiente sección.

El código completo y comentado para esta clase se puede consultar en el Anexo correspondiente.

V.3 Las clases concretas Componente1T, Componente2T, Alambre Corto2T y Componente3T

Estas **clases concretas** extienden a la **clase abstracta Componente**. La **propiedad datosComponente** explicada en la sección anterior, también se extiende en cada una de estas clases concretas, como se muestra en la Figura V.15.

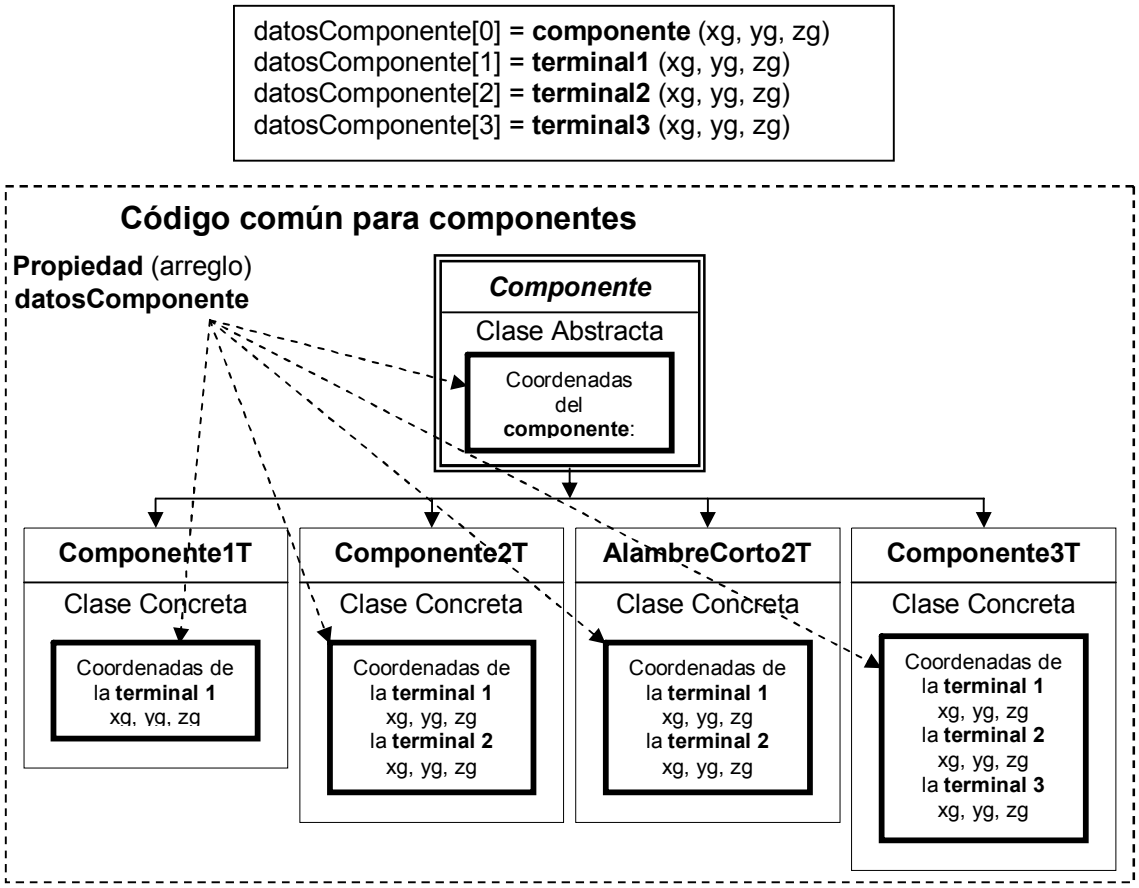


Figura V.15. Estructura de la **propiedad datosComponente** distribuida entre la **clase abstracta Componente** y las **clases concretas** que la **extieneden**.

La clases concretas anteriores están estructuradas con las **propiedades** y **métodos** que son comunes a todos los componentes de una, dos o tres terminales (respectivamente). En las Figuras V.16 y V.17, se muestran las estructuras generales del código de estas clases concretas.

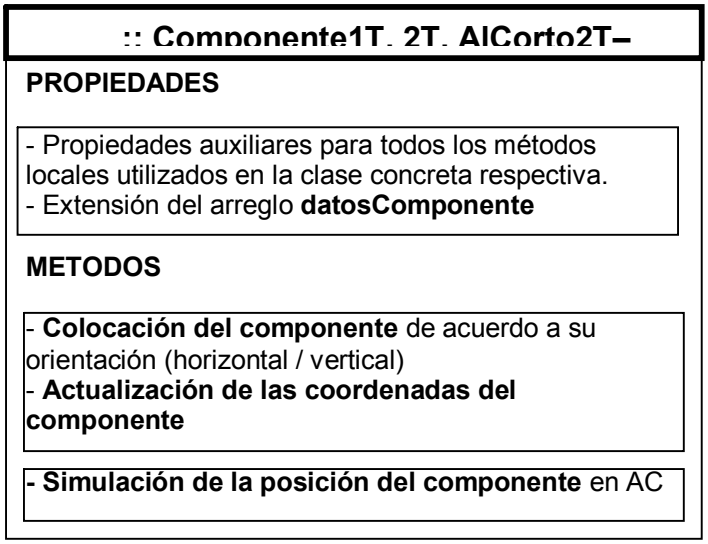


Figura V.16. Estructura del código general de las clases concretas **Componente1T**, **Componente2T** y **AlambreCorto2T**.

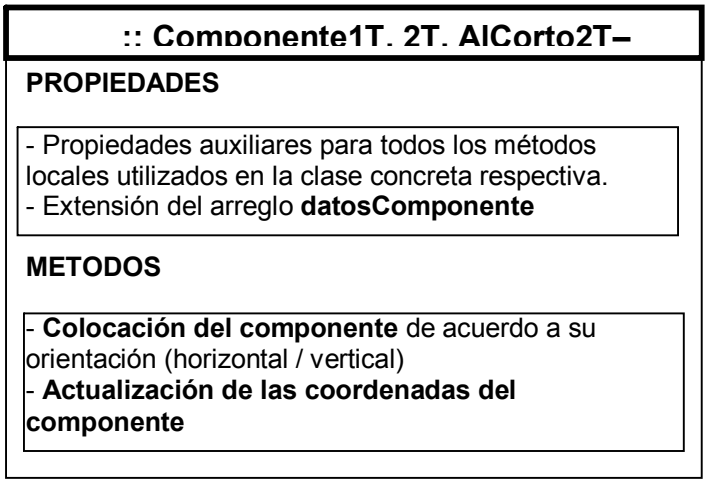


Figura V.17. Estructura del código general de la clase concreta **Componente3T**.

Como se observa en las Figuras V.16 y V.17, los métodos definidos para las clases concretas **Componente1T**, **Componente 2T**, **AlambreCorto2T** y **Componente 3T**, se encargan de colocar el componente con la **orientación** (vertical / horizontal) solicitada y actualizar las coordenadas del arreglo **datosComponente** de acuerdo a dicha orientación. Solamente las tres primeras clases poseen un método encargado de la **simulación de la posición del componente en AC**. En la clase **Componente3T** este método no está presente en dicha clase, sin embargo si está definido en la clase particular del componente de tres terminales, en este caso el **Amplificador Operacional** (clase **OpAmp**).

El código completo y comentado para estas clases se puede consultar en el Anexo correspondiente.

V.4 La clase ventanaEdicion

Uno de los métodos de la clase abstracta **Componente** descrita en la sección V.1.5, tiene la función de generar una **ventana de edición** que permita **editar algunos de los parámetros de los componentes** que se utilizan en el dibujo de un circuito en el **Area de Dibujo**. La clase **Componente** crea esta ventana para cada componente, como se observa en la Figura V.17.

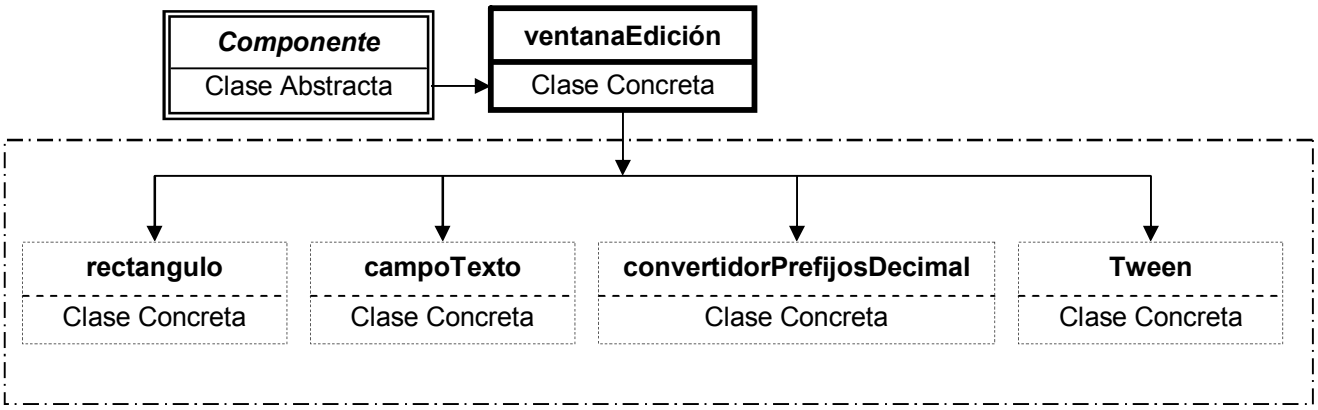


Figura V.17. Diagrama de conexión de la clase **ventanaEdicion** con la **clase abstracta Componente** y con sus clases auxiliares: **rectangulo**, **campoTexto**, **convertidorPrefijoDecimal** y **Tween**.

Esta clase genera una **ventana de edición** para cada componente, cada vez que se realiza un “doble click", con el puntero del **mouse**, sobre algún componente. En la Figura V.18 se muestran varias **ventanas de edición abiertas** para cada **componente** de un circuito de ejemplo.

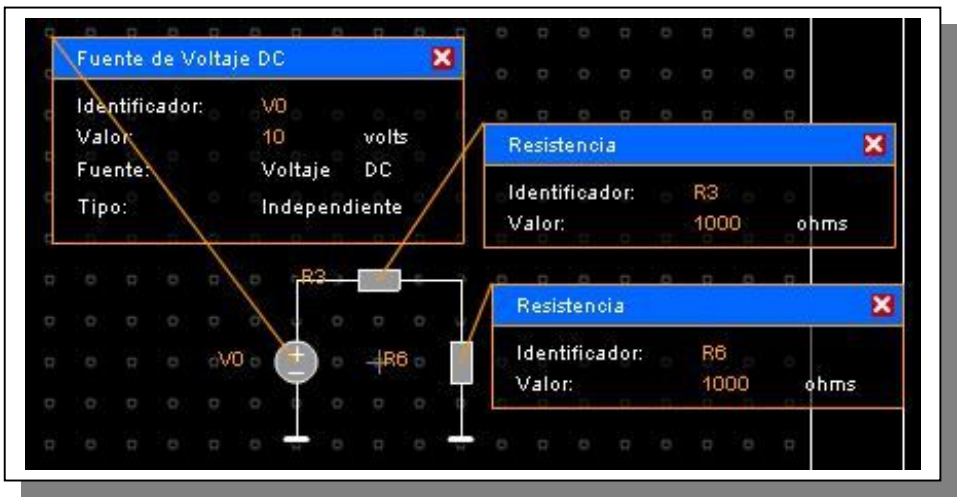


Figura V.18. **Ventanas de edición abiertas** para cada componente de un circuito de ejemplo.

En la parte superior de dichas ventana aparece el **nombre del componente** y un cuadro en el extremo derecho que permite **cerrar la ventana**. Todas las ventanas abiertas deberán cerrarse antes de realizar cualquier simulación del circuito, ya sea en DC o en AC. A continuación, en cada ventana, aparece información particular del componente al cual esta conectada dicha ventana mediante una línea. Solamente la información que aparece en color "naranja" dentro de la **ventana de edición**, podrá editarla (alterarla) el usuario. Las palabras o números que aparecen en color blanco dentro de la ventana, solo son informativos y no pueden editarse.

Las ventanas de edición abiertas se pueden **desplazar** con el mouse para colocarlas en cualquier parte de la pantalla. Para hacer esto, basta con colocar el puntero del mouse sobre la franja superior de la ventana y arrastrarla presionando el botón izquierdo del mouse. Como se observa en la figura anterior, también pueden estar abiertas varias ventanas al mismo tiempo.

La estructura del código de la clase **ventanEdición** se muestra en la Figura V.19.

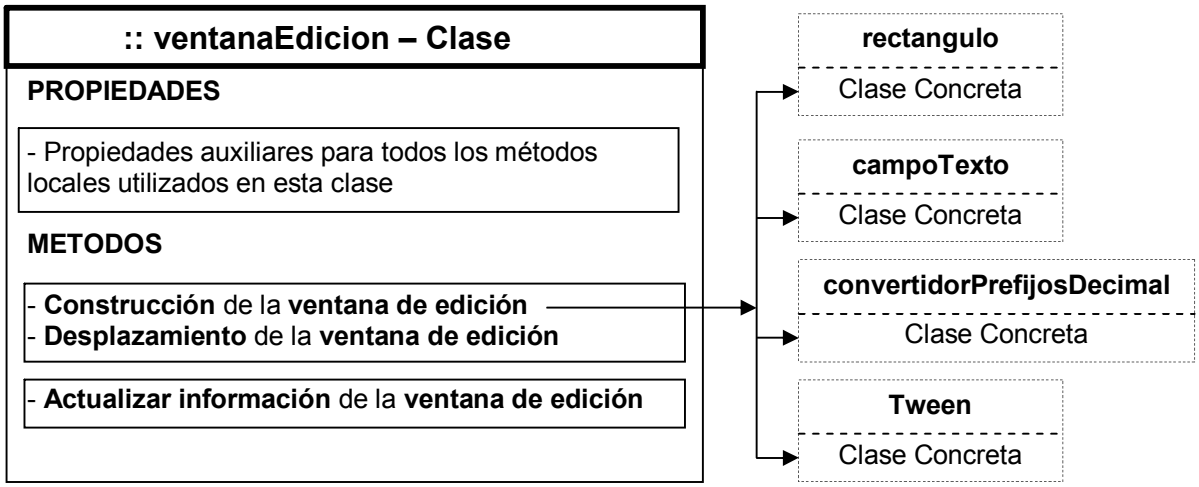


Figura V.19. Estructura del código de la clase concreta **ventanaEdicion**.

Los **métodos** que aparecen en el código de la figura, constituyen dos grupos de funciones comunes a todos los componentes, cuyas funciones son:

1. **Construir la ventana de edición** mediante el dibujo de un **rectángulo** y la **definición de campos de texto estáticos y de entrada**. El formato permitido en el campo de texto de entrada será de **notación decimal** y **notación con prefijos**, por lo que se requiere del auxilio de la clase **convertidorPrefjoDecimal**. Finalmente, la ventana se abre aumentando gradualmente su tamaño y se cierra disminuyendo, también, gradualmente su tamaño, lo cual se logra mediante la clase **Tween**.

2. El método de **actualización de información** es un método auxiliar utilizado por la clase externa **manejadorCircuito**, la cual se explicará mas adelante, en la sección correspondiente a la estructura **MVC**.

El código completo y comentado para esta clase se puede consultar en el Anexo correspondiente.

Las clases auxiliares **rectángulo** y **campoTexto**, como su nombre lo indica se encargan de dibujar un rectángulo y de definir y manejar campos de texto estáticos (no editables por el usuario) y de entrada (editables). Por su simplicidad no se muestran aquí las estructuras de código de estas clases, sin embargo, el código completo y comentado se puede consultar en el Anexo correspondiente.

La clase auxiliar **convertidorPrefjoDecimal** se encarga de convertir determinados prefijos a un número decimal, con la finalidad de que los pueda manejar el simulador, el cual también se explicará mas adelante en la sección de la estructura **MVC**. Los prefijos que convierte esta clase se muestran en la Figura V.20.

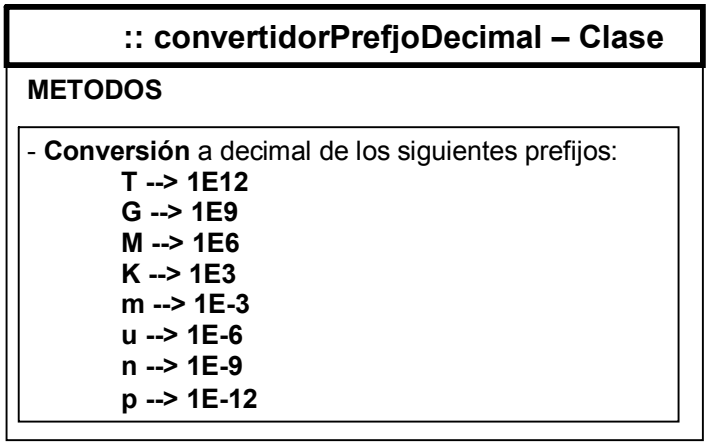


Figura V.20. Estructura del código de la clase concreta **convertidorPrefjoDecimal**.

La conversión de los prefijos se realiza mediante un análisis de la cadena de caracteres escrita por el usuario. El código completo y comentado para esta clase se puede consultar en el Anexo correspondiente.

V.5 Las clases de los componentes

En la Figura V.21 se muestra gráficamente un listado clasificado de las **clases** para los componentes que se pueden manejar en el simulador.

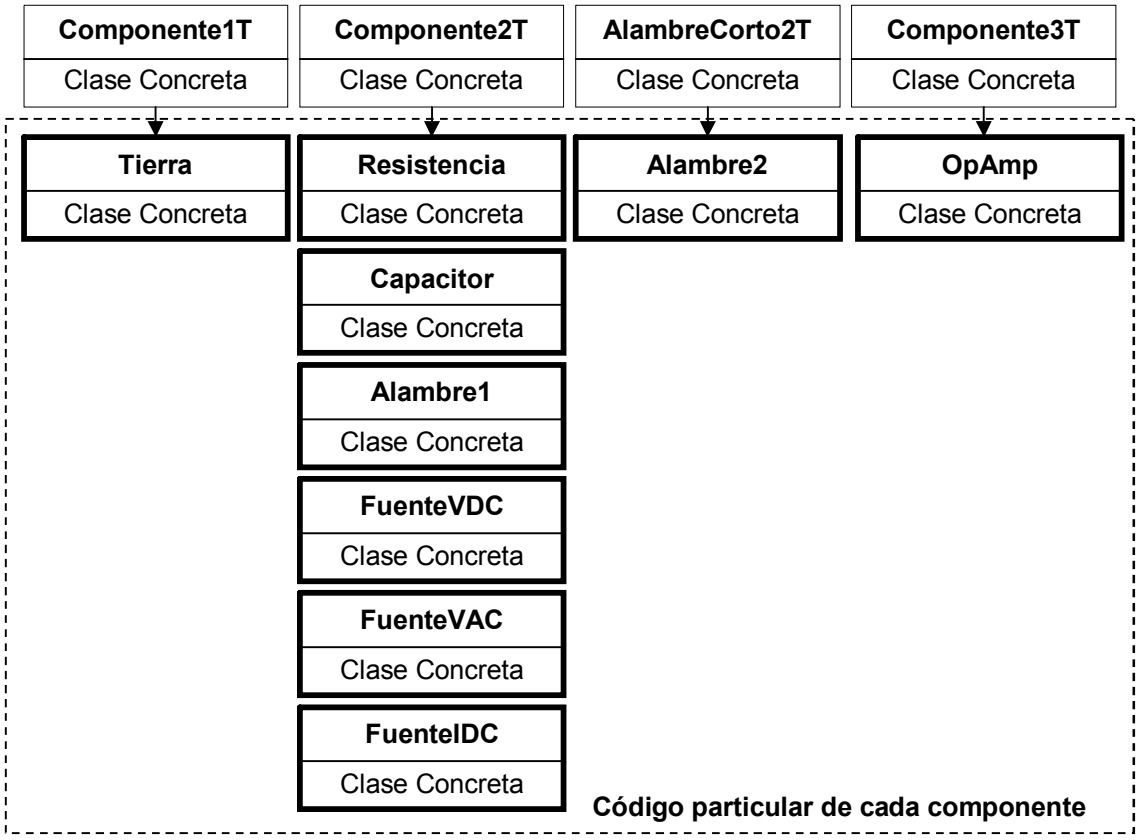


Figura V.21. Listado clasificado de las **clases** para los componentes que se pueden manejar en el simulador.

Los nombres de las **clases** describen por si solos el tipo de **objeto componente** que pueden generar. La estructura del código de cada clase comparte el formato común que se muestra en la Figura V.22.

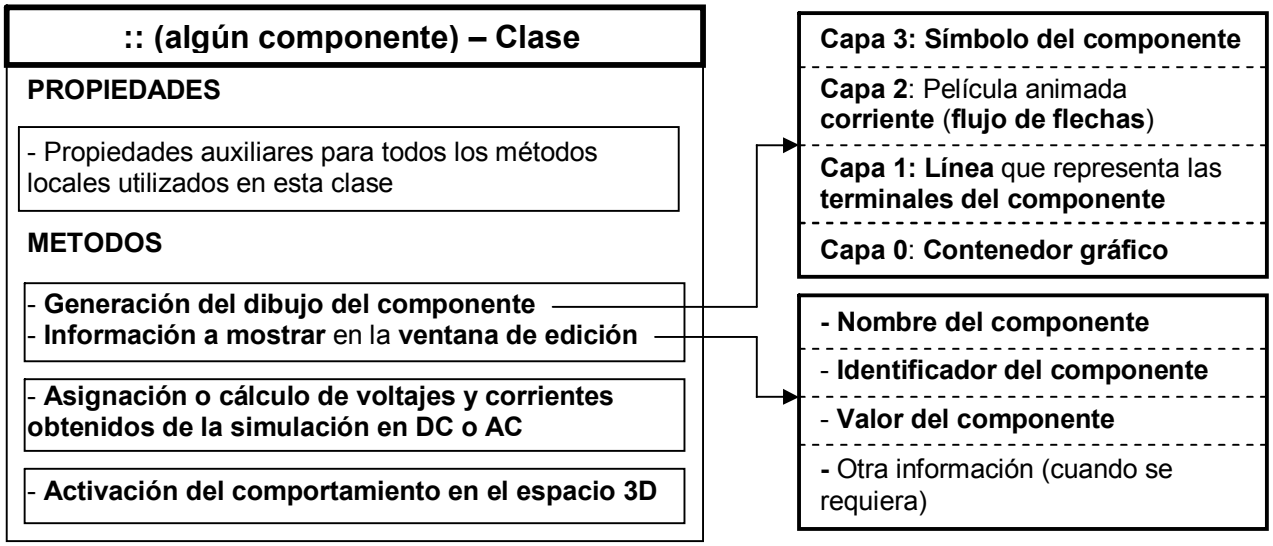


Figura V.22. Estructura común del código de las clases concretas de cada **componente** en particular.

Los **métodos** que aparecen en el código de la figura, constituyen tres grupos funcionales comunes a todos los componentes, cuyas funciones son:

1. **Generar el dibujo del componente**. – El dibujo de cada componente se construye mediante **cuatro capas**, como se ejemplifica en la Figura V.23. Esta separación de los componentes del símbolo en **cuatro capas**, es necesaria para lograr proyectarlo en el espacio 3D sin deformarlo, además de permitir la animación del mismo

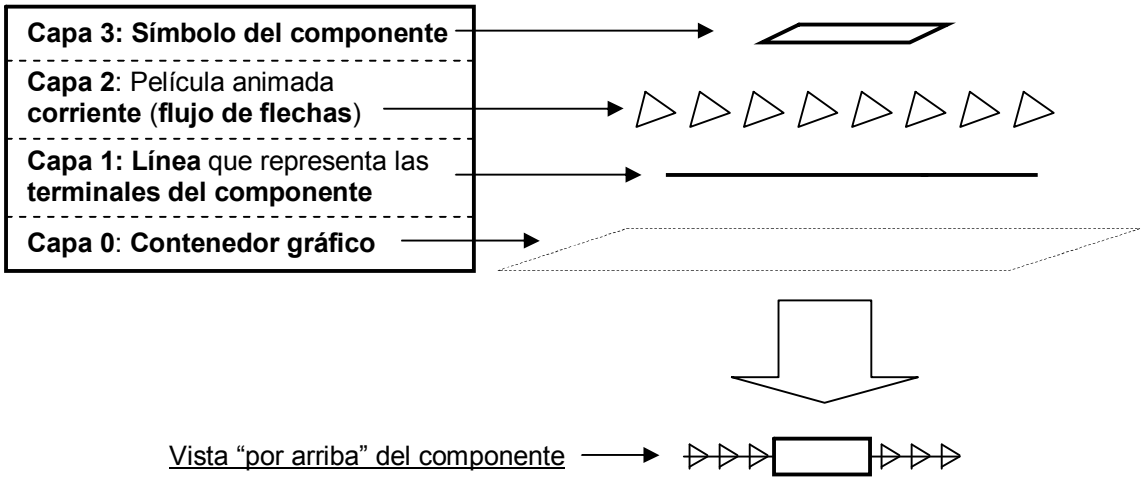


Figura V.23. Ejemplo de la construcción del **símbolo** de una resistencia mediante una **estructura de cuatro capas**.

2. Definir la **información a mostrar en la ventana de edición del componente**. – Esta información se encuentra en cadenas de caracteres y/o números guardados en un arreglo de cadenas de caracteres llamado **infoVentana**, el cual lo consulta la clase **ventanaEdición** y lo despliega en su ventana correspondiente al componente que tiene asociado. La estructura de este arreglo es la que se muestra en la Figura V.24.

- Nombre del componente
- **Identificador del componente**
- **Valor del componente**
- Otra información (cuando se requiera)

Figura V.24. Estructura del arreglo **infoVentana**.

3. **Asignar o calcular los voltajes nodales y/o las corrientes de rama resultantes de la simulación del circuito en DC o AC**. – Estos resultados o cálculos se guardan en las **PROPIEDADES DE CLASE** de cada componente en particular, las cuales serán consultadas por el bloque **Vista** (de la estructura **MVC**) para su despliegue en las gráficas correspondientes.

4. **Activación del comportamiento en el espacio 3D**. – Consiste en colocar el componente en el espacio 3D y calcular su orientación y comportamiento en dicho espacio. La Figura V.25 muestra un ejemplo del símbolo de una resistencia colocado y orientado un cierto ángulo en el espacio 3D. El plano inferior representa el plano de referencia o de potencial 0 volts.

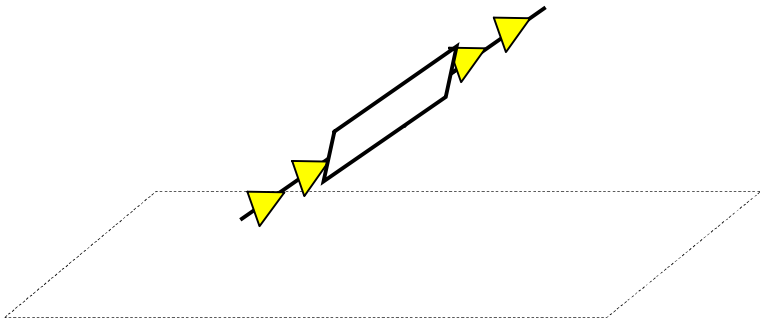
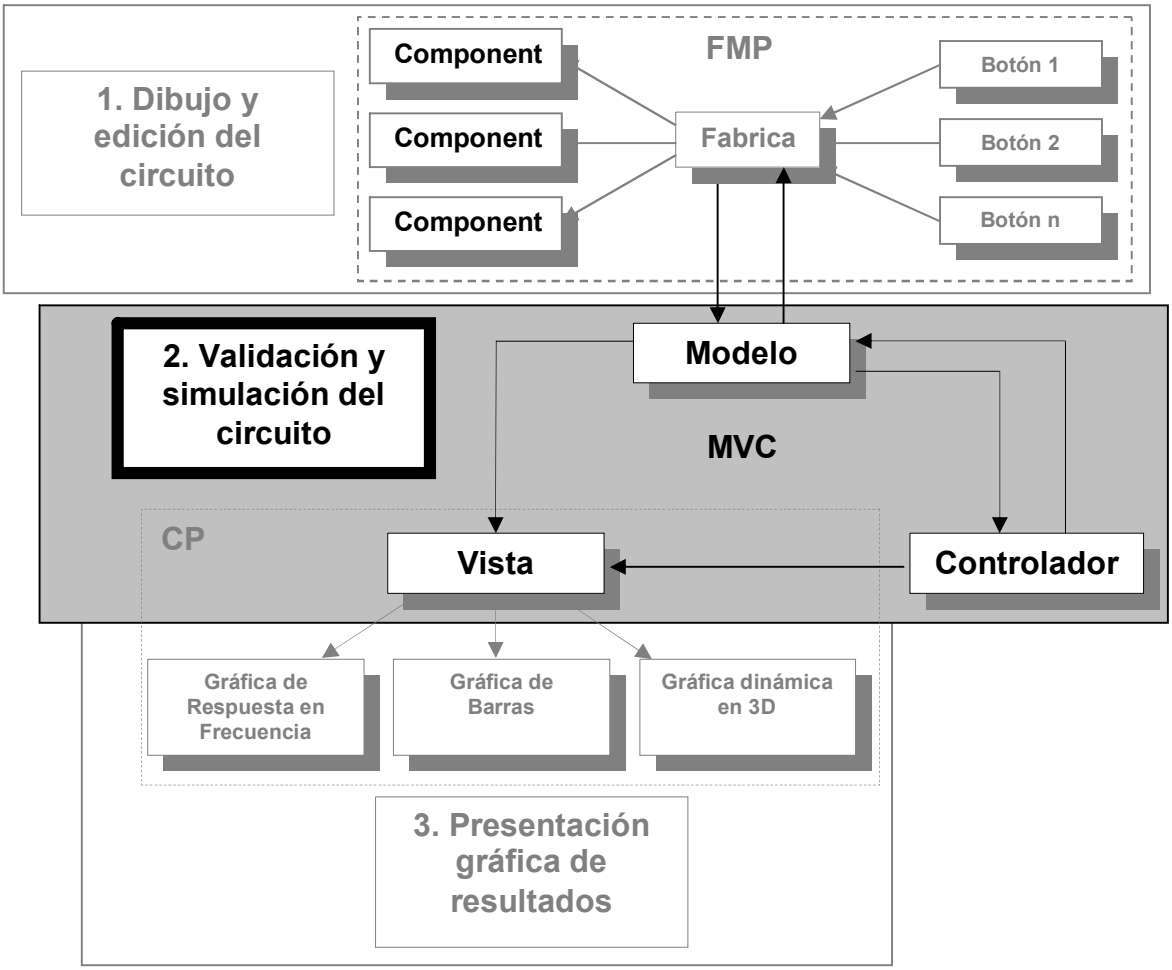


Figura V.25. Colocación del **símbolo** de una resistencia en el espacio 3D.

Se hace notar que la sección de código de clase llamado **simulación de la posición del componente en AC**, reside en la clases **Componente1T**, **Componente2T** y **AlambreCorto2T**. Para el caso de la clase **Componente3T**, dicho código no reside en esta clase sino en la clase particular del componente de

tres terminales (el Amplificador Operacional en este caso). Esta sección de código mencionada, se encarga, junto con la clase **motor_3D_ac** (correspondiente a la sestructura **MVC**), de **"oscilar**" al componente alrededor de su punto de registro de acuerdo a los resultados de la **simulación en AC** (magnitud y fase).

El código completo y comentado para todas las **clases** de cada componente se puede consultar en el Anexo correspondiente.



VI. Patrón de Programación MODELO - VISTA – CONTROLADOR (MVC).

El patrón de programación **MVC** se encuentra asociado con el manejo de la red del circuito, la simulación del mismo y la representación de resultados en forma gráfica.

La función de esta estructura de programación es la de **consultar el arreglo de objetos gráficos** generado por la estructura **FMP** presentada en el

capítulo anterior, y **procesarlo** hasta generar el tipo de datos requerido para la representación de éstos en **gráficas 2D** y **3D**.

El procesamiento del **arreglo de objetos gráficos** por parte de esta estructura se divide en tres etapas, como se muestra en la Figura VI.1.

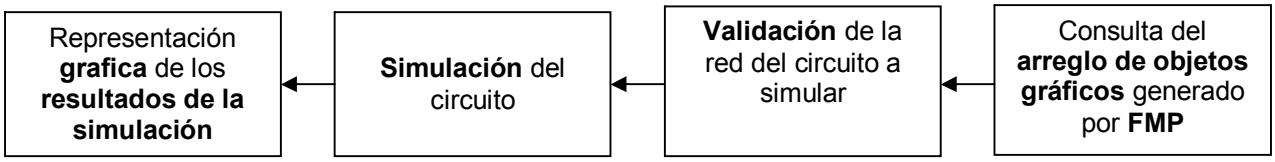


Figura VI.1. Procesamiento del **arreglo de objetos gráficos** que representa el circuito a simular.

El procesamiento comienza con la consulta del **arreglo de objetos gráficos** generados por la estructura **FMP**. Se **valida** la red resultante que constituye una descripción del circuito, se **simula** y, finalmente, los resultados obtenidos se grafican en **gráficas 2D y 3D**.

VI.1 Diagrama de clases

En al Figura VI.2, se muestra en forma gráfica la arquitectura detallada del patrón de diseño de programación **MVC** utilizado en el proyecto. Esta estructura constituye una adaptación de dicho modelo a las necesidades del simulador. En la figura se indican las conexiones y el flujo de información entre cada uno de los elementos que lo constituyen.

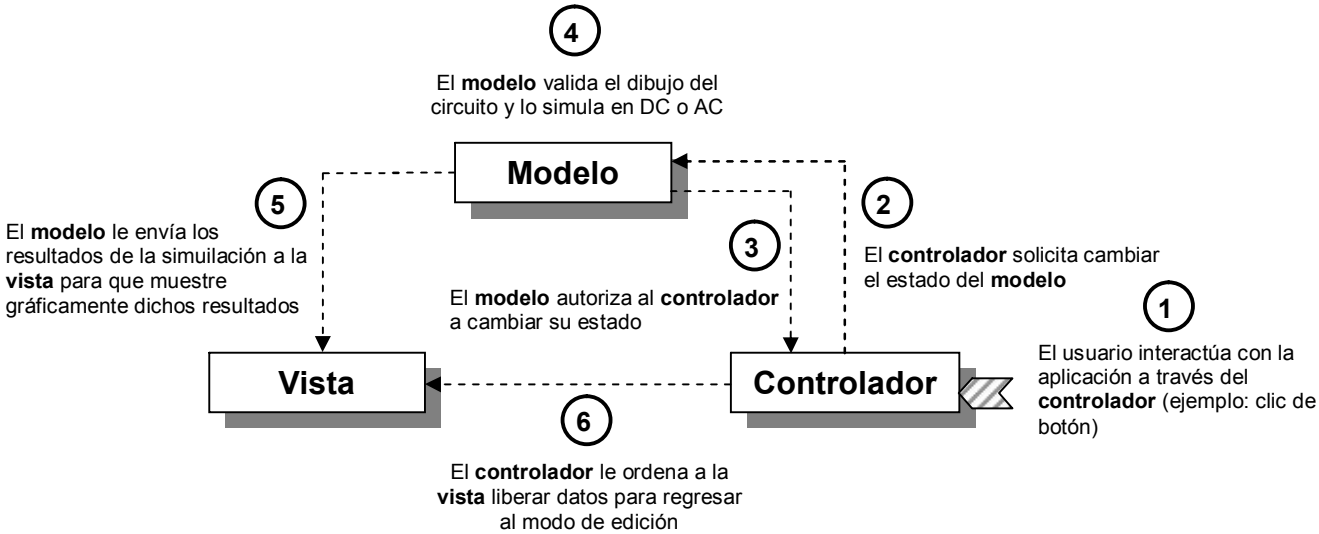


Figura VI.2. Arquitectura y flujo de información entre los elementos del patrón de programación **MVC**

Cada uno de los bloques del sistema **MVC** realizará las siguientes funciones:

- **Controlador.** – Controla el flujo global del simulador, previa autorización del **Modelo**, controlando los tres botones del Menú Principal: **EDICION**, **SIMULAR EN DC** y **SIMULAR EN AC**.
- **Modelo** – Controla el estado del sistema, analiza y valida el dibujo del circuito, genera un modelo matemático del circuito y lo simula en **DC** o **AC**.
- **Vista** – Crea los tres tipos de gráfica para mostrar los resultados de la simulación: **gráfica de barras**, **grafica de respuesta en frecuencia** (**AC**) y **graficas dinámicas en 3D en DC y AC**.

Cada módulo mostrado en la figura anterior, representa una "clase". Los objetos correspondientes a cada "clase", así como la conexión entre dichos objetos, se generan en una **clase inicial** del proyecto llamada **Main**, la cual se encarga de generar los objetos y conectarlos entre sí de acuerdo al diagrama establecido en la Figura VI.2 y cuyo código en ActionScript 3, se muestra en la Figura VI.3

```
:: Main
CONSTRUCTOR
// --- Modelo ---
var modelo:ModeloCircuito = new ModeloCircuito(this);
// --- Vistas ---
var muestraResultadosDC:vistasDC = new vistasDC(modelo);
var muestraResultadosAC:vistasAC = new vistasAC(modelo);
// --- Controlador ---
var cont:controlador = new controlador(modelo, muestraResultadosDC,
                                       muestraResultadosAC);
```

Figura VI.2. Código de la clase principal **Main** que se encarga de generar y conectar los objetos de acuerdo al modelo **MVC** mostrado en el diagrama de la Figura VI.1

La creación de los objetos en el código anterior, se realiza mediante el comando **new**. La conexión entre los objetos se realiza a través de los argumentos de las funciones **constructoras** de las **clases** correspondientes.

Los **dos objetos vistas** (**DC** y **AC**), se conectan con el **objeto modelo** a través de su argumento del **constructor** correspondiente.

El **objeto controlador** se conecta con el **objeto modelo** y los **dos objetos vistas**, también a través del argumento de su **función constructora**.

Con este código, se crean las **conexiones** entre los objetos como se muestra en la Figura VI.2. Sin embargo, la **dirección** del flujo de información entre dichas conexiones, se encuentra definida en los **métodos de acceso** de cada **clase**.

VI.2 La clase Modelo

La clase **Modelo** está estructurada con las **propiedades** y **métodos** que se muestran en la Figura VI.3.

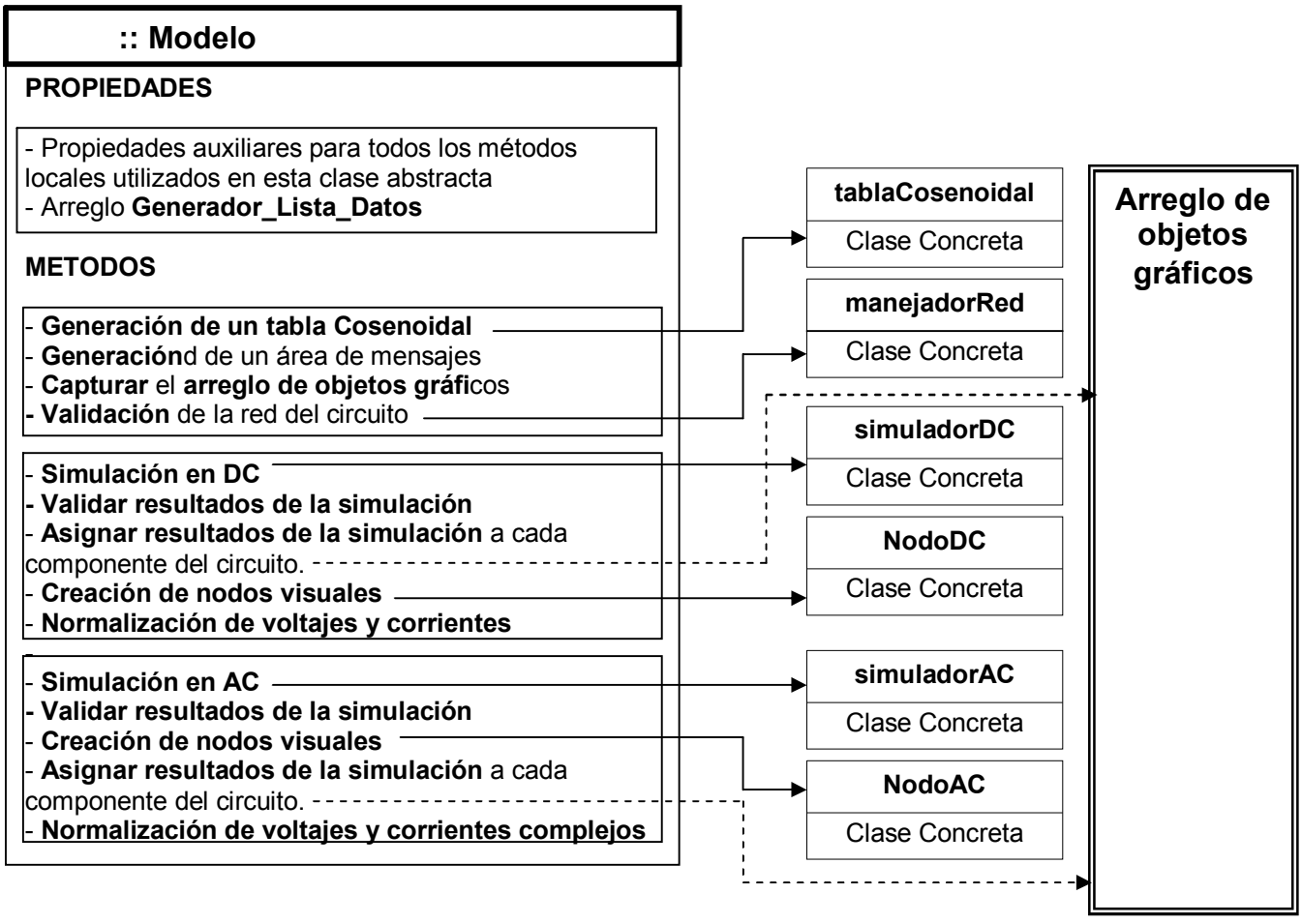


Figura VI.3. Estructura del código de la clase **Modelo**.

En la **propiedad** de la clase llamada **Generador_Lista_Datos** se guardan todo los valores y resultados asociados con la **simulación en AC**

Los **métodos** que aparecen en el código de la figura, se encuentran calsificados en tres grupos, cuyas funciones son:

1. Generar una **tabla cosenoidal** para utilizarse en la generación del movimiento oscilatorio de los componentes en la **simulación en AC.** Generar un **área de mensajes** para el usuario. **Capturar y validar la red del circuito** que se va a simular.
2. **Simular el circuito en DC**. **Validar y asignar resultados de la simulación a todos los componentes** que forman el circuito. **Crear nodos visuales (DC)**, es decir, dibujos de nodos que representen gráficamente el comportamiento de un nodo del circuito. **Normalizar los valores de voltaje y corriente** para representarlos siempre dentro de los límites de graficación del proyecto. Los límites para el voltaje son 160 pixeles máximo de diferencia de altura entre dos nodos, y para la corriente un valor de 10 pixeles como anchura máxima del flujo de flechas.
3. **Simular el circuito en AC**. **Validar y asignar resultados de la simulación a todos los componentes** que forman el circuito. **Crear nodos visuales (AC)**, es decir, dibujos de nodos que representen gráficamente el comportamiento de un nodo del circuito. **Normalizar los valores de voltaje y corriente** para representarlos siempre dentro de los límites de graficación del proyecto. Los límites para el voltaje son 160 pixeles máximo de diferencia de altura entre dos nodos.
4. La **asignación de los resultados de la simulación** se realiza directamente en cada componente del circuito a través de las referencias guardadas en el **arreglo de objetos gráfico**.

La **propiedad** de la clase llamada **Generador_Lista_Datos**, constituye una clase utilizada para generar listas de valores utilizado exclusivamente en la **simulación en AC** del circuito. También aquí se guardan todos los resultados obtenidos en la simulación. Esta clase se utiliza como un almacén de datos debido a la gran cantidad de información que se genera, ya que en este tipo de simulación se realiza un barrido en frecuencia de 400 muestras (400 frecuencias diferentes), y los resultados complejos de los voltajes se guardan en dos formatos: complejo y polar para cada frecuencia.

El código completo y comentado para esta clase se puede consultar en el Anexo correspondiente.

VIII. Resultados

Tomando el simulador PSPICE como referencia de uno de los simuladores mas utilizados a nivel universidades e industrias, se presentan en forma comparativa las novedades que presenta el proyecto realizado. Para fines de comparación, el simulador realizado en el proyecto se le llamará Sim3D.

VIII.1 El editor de circuitos

El editor de PSPICE al igual que el del proyecto, permiten dibujar un circuito plano en 2D, como se muestra en la Figura VIII.1. El editor del proyecto no tiene comandos para rotar componentes como los que tiene PSPICE, por lo que es mas fácil su uso.

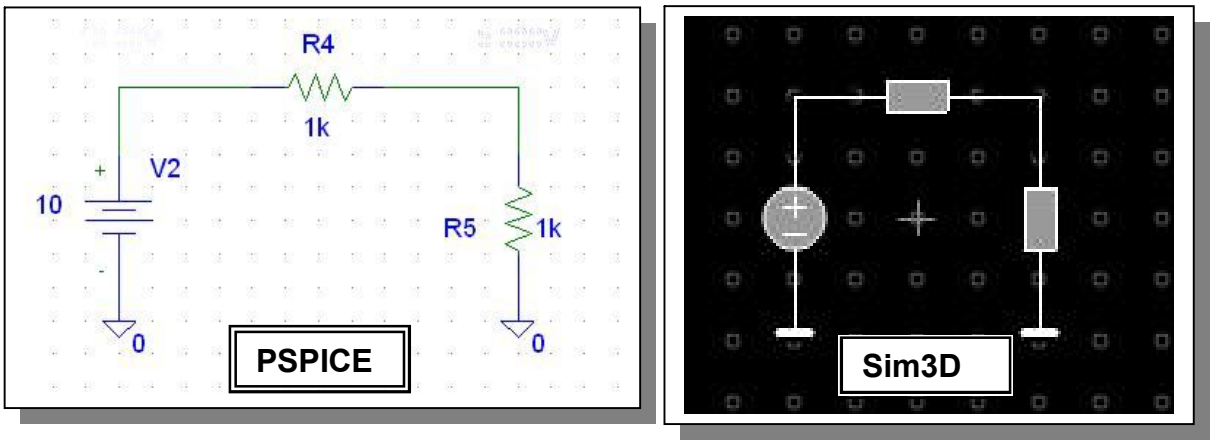


Figura VIII.1 Editores de PSPICE y del Sim3D

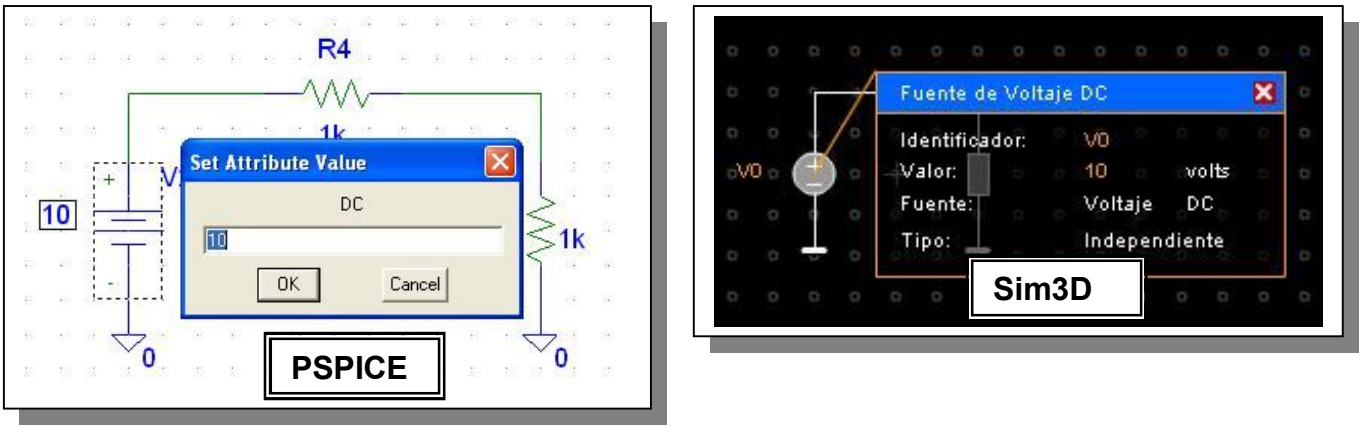


Figura VIII.2 Edición de parámetros de los componentes en PSPICE y del Sim3D

VIII.2 Simulación

El simulador Sim3D utiliza el Método Nodal Modificado para la simulación de los circuitos, el cual genera resultados iguales a los realizados mediante un análisis manual o a través del simulador PSPICE, como se muestra en la Figura VIII.3

VIII.3 Presentación de resultados

El simulador Sim3D presenta todos sus resultados en forma gráfica e interactiva, como se muestra en la misma Figura VIII.3.

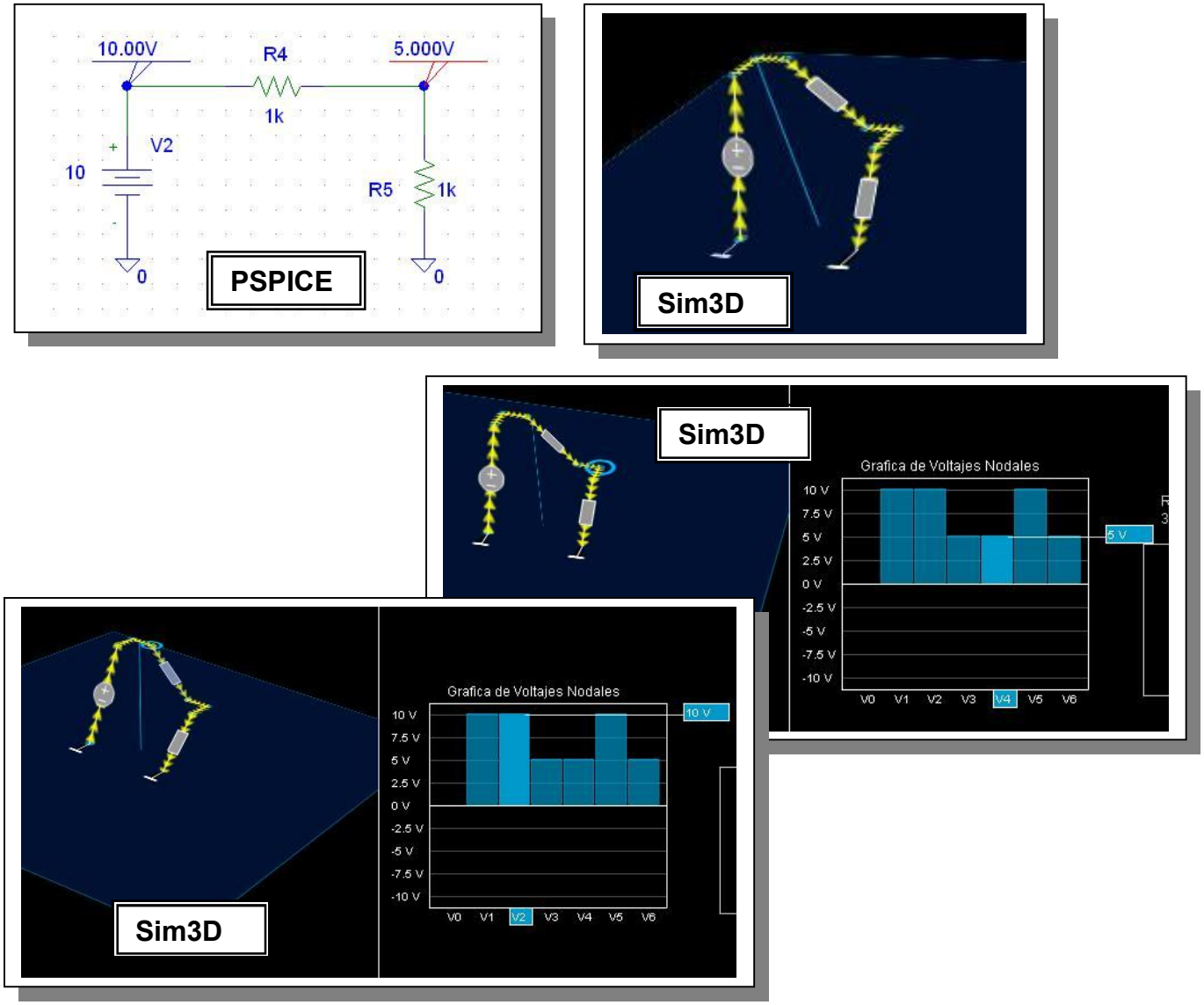


Figura VIII.3 Simulación y presentación de resultados en PSPICE y en Sim3D

IX. CONCLUSIONES

De acuerdo a los objetivos del proyecto, puedo afirmar que se lograron alcanzar.

En cuanto a la funcionalidad del proyecto, se lograron conjuntar tres cosas: un editor gráfico sencillo, un simulador de circuitos eléctricos y electrónicos y la representación de los resultados del simulador en un gráfica del circuito en 3D.

En cuanto a la construcción del proyecto, se utilizó como estructura de programación base para la realización del mismo, el patrón de programación MVC (Modelo – Vista – Controlador), además de que fue necesaria la adición de un patrón de programación mas, el FMP (Método de la Fábrica). El patrón de programación CP (Composición), también utilizado en el proyecto, se encuentra traslapado en el módulo Vista del modelo MVC.

El editor resultó muy sencillo de usar y muy intuitivo, ya que carece de los comandos de rotación de componentes como el que presenta el editor de PSPICE, lo cual le da gran facilidad de uso.

El simulador, por estar basado en el Método Nodal Modificado, genera resultados iguales a los realizados mediante un análisis manual o a través del simulador PSPICE para un determinado circuito.

La gráficas en 2D y 3D que se obtuvieron son de buena calidad, así como el comportamiento de las mismas a la interactividad de éstas con el usuario.

Sería recomendable intentar otros formatos para las gráficas en 3D, como pueden ser, crear efectos de mayor iluminación de los componentes que estén más cerca del usuario; incluir la función de zoom; crear componentes en 3D en lugar de componentes en 2D, como los que se utilizaron; incrementar la información proporcionada por las gráficas (potencia, energía, etc).

BIBLIOGRAFÍA

[1] ActionScript 3.0 Design Patterns
William Sanders and Chandima Cumaranatunge
Ed. O´REILLY, ADOBE, 2007

[2] Introducción a PSPICE
James W. Nilsson and Susan A. Riedel
Ed. Addison-Wesley, 1994

[3] Análisis y Diseño de Circuitos con Computadora
Joan María Miró Sans, Antonio Puerta Notario,
José María Miguel López, Margarita Sanz Postilla
Ed. Alfaomega - Marcombo, 1989

[4] Programming ADOBE ACTIONSCRIPT 3.0
ADOBE, 2008

[5] ActionScript: Making Things Move
Keith Peters
Ed. friendsofED, 2006

[6] Advanced ActionScript 3.0 Animation
Keith Peters
Ed. friendsofED, 2009

[7] ActionScript 3.0: Bible
Roger Braunstein, Mims H. Wright and Joshua J. Noble
Ed. Wiley, 2008

[8] ActionScript 3.0 Cookbook
Joey Lott, Darron Schall and Keith Peters
Ed. O´REILLY, ADOBE, 2007

[9] ActionScript 3.0 Game Programming University
Gary Rosenzweig´s
Ed. QUE, 2008

[10] Programming Macomedia Flash MX
Robert Penner´s
Ed. OSBORNE, McGrawHill, 2002

[11] Flash Math Creativity
Varios Autores
Ed. friendsofED, 2004

[12] Flash and Math Applets: Learn by Example
Douglas Ensley and Barbara Kaskosz
Ed. Ensley and Kaskosz, 2009

[13] Fundamentos de Circuitos Eléctricos
Charles K. Alexander, Mattew N. O. Sadiku
Ed. McGrawHill, 2006

[14] Circuitos Eléctricos: Introducción al Análisis y Diseño
Richard C. Dorf
Ed. Alfaomega, 1995

[15] Active Filter Opamp Cookbook
Don Lancaster
Ed. SAMS, 1985

[16] La sintaxis de la imagen: Introducción al alfabeto visual
D. A. Dondis
Ed. Gustavo Gil, 2000

[17] Bases del Diseño Gráfico
Alan Swann
Ed. Gustavo Gil, 2006

[18] Fundamentos del Diseño
Wucius Wong
Ed. Gustavo Gil, 2007

[19] Ohm Zone
http://www.article19.com/shockwave/oz.htm

[20] ActionScript 3.0 Tutorials
http://www.flashandmath.com/

```
package com.fabrica.consumidor.registroBotones
{
	// ************************************************************
	//
	//		Clase:	CrearBotonesComponentes (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Botones de componentes
	import com.fabrica.consumidor.botonesComponentes.resistencia.botonResistencia;
	import com.fabrica.consumidor.botonesComponentes.capacitor.botonCapacitor;
	import
com.fabrica.consumidor.botonesComponentes.fuenteVoltajeDC.botonFuenteVoltajeDC;
	import
com.fabrica.consumidor.botonesComponentes.fuenteCorrienteDC.botonFuenteCorrienteDC;
	import
com.fabrica.consumidor.botonesComponentes.fuenteVoltajeAC.botonFuenteVoltajeAC;
	import com.fabrica.consumidor.botonesComponentes.alambre.*;
	import com.fabrica.consumidor.botonesComponentes.tierra.botonTierra;
	import
com.fabrica.consumidor.botonesComponentes.amplificador_operacional.botonOpAmp;

	public class CrearBotonesComponentes extends Sprite {

		// ---------------------------------
		//                                COLORES
		// ---------------------------------
		public var verde:Number = 0x00ff99;
		public var naranja:Number = 0xff6633;
		public var naranja2:Number = 0xff9900;
		public var azul:Number = 0x0099ff;
		public var azul2:Number = 0x0066ff;
		public var amarillo:Number = 0xffff00;
		public var violeta:Number = 0xff00ff;
		public var rojo:Number = 0xff0000;
		public var rojo2:Number = 0xcc0000;
		public var cafe:Number = 0xcc6633;

		public var blanco:Number = 0xffffff;
		public var gris:Number = 0x666666;
		public var negro:Number = 0x000000;

		// --- Posicion de botones ---
```

```
		public var posicionBoton:Array;
		public var posicionInicial:uint = 10;
		public var tamanioBoton:uint = 50;
		public var separacion:uint = 2;

		// --- Datos ---
		public var spMenuComponentes:Sprite;
		public var spEditor:Sprite;
		public var parametros:Object

		public function CrearBotonesComponentes(spMenuComponentes:Sprite,

	spEditor:Sprite,

	parametros:Object):void {

			// --- Recepcion de datos ---
			this.spMenuComponentes = spMenuComponentes;
			this.spEditor = spEditor;
			this.parametros = parametros;

			// --- Genera posisiones de Botones en 8 filas y 6 columnas ---
			generaPosicionesBotones();
			// --- Crear botones de componentes ---
			generarBotonesComponentes();
		}

		public function generarBotonesComponentes():void {

			// ------------------------------------------------------------
			// Estructura de Datos del argumento "arg" para solicitar
			// la creacion de un boton:
			//
			// +++ Constantes +++
			// arg.spMenuComponentes --> "sprite" en donde se colocan
			//                                                    los botones
			// arg.spEditor                --> "sprite" en donde se colocan
			//                                                    los componentes
			// arg.generaCoordenadas --> "objeto" parametros
			// +++ Variables +++
			// arg.fila y arg. columna > coordenadas para colocacion
			//                                                    del boton
			// arg.color                   --> color del "marco" del boton
			// arg.orientacion             --> orientacion del "logo" del boton
			//                                                    y del componente:
			//                                                    "W" <- | "N" ^ | "E"
-> | "S" v
			// ------------------------------------------------------------
			var arg:Object = new Object();

			// --- Recepcion de datos ---
			// --- Argumentos constantes ---
			arg.spMenuComponentes = spMenuComponentes;
			arg.spEditor = spEditor;
			arg.generaCoordenadas = this;
			arg.parametros = parametros;
```

```
                        // --- Creacion de botones de componentes ---
                        // --------------------------------
                        //          FILA: 0
                        // --------------------------------
                        //   FUENTES DE VOLTAJE DC
                        // --------------------------------
                        arg.fila = 0;
                        arg.color = azul;
                        // --- BOTONES >FUENTE DE VOLTAJE DC: Argumentos variables ---
                        arg.columna = 0;
                        arg.orientacion = "N";
                        var btnVDC1:botonFuenteVoltajeDC = new botonFuenteVoltajeDC(arg);

                        arg.columna = 1;
                        arg.orientacion = "S";
                        var btnVDC2:botonFuenteVoltajeDC = new botonFuenteVoltajeDC(arg);

                        arg.columna = 2;
                        arg.orientacion = "E";
                        var btnVDC3:botonFuenteVoltajeDC = new botonFuenteVoltajeDC(arg);

                        arg.columna = 3;
                        arg.orientacion = "W";
                        var btnVDC4:botonFuenteVoltajeDC = new botonFuenteVoltajeDC(arg);

                        // --------------------------------
                        //          FILA: 1
                        // --------------------------------
                        //   FUENTES DE CORRIENTE DC
                        // --------------------------------
                        arg.fila = 1;
                        arg.color = azul;
                        // --- BOTONES >FUENTE DE VOLTAJE DC: Argumentos variables ---
                        arg.columna = 0;
                        arg.orientacion = "N";
                        var btnIDC1:botonFuenteCorrienteDC = new
botonFuenteCorrienteDC(arg);

                        arg.columna = 1;
                        arg.orientacion = "S";
                        var btnIDC2:botonFuenteCorrienteDC = new
botonFuenteCorrienteDC(arg);

                        arg.columna = 2;
                        arg.orientacion = "E";
                        var btnIDC3:botonFuenteCorrienteDC = new
botonFuenteCorrienteDC(arg);

                        arg.columna = 3;
                        arg.orientacion = "W";
                        var btnIDC4:botonFuenteCorrienteDC = new
botonFuenteCorrienteDC(arg);

                        // --------------------------------
                        //          FILA: 2
```

```
// ---------------------------------
//   FUENTES DE VOLTAJE AC
// ---------------------------------
arg.fila = 2;
arg.color = azul;
// --- BOTONES >FUENTE DE VOLTAJE AC: Argumentos variables ---
arg.columna = 0;
arg.color = azul;
arg.orientacion = "N";
var btnVAC1:botonFuenteVoltajeAC = new botonFuenteVoltajeAC(arg);

arg.columna = 1;
arg.orientacion = "S";
var btnVAC2:botonFuenteVoltajeAC = new botonFuenteVoltajeAC(arg);

arg.columna = 2;
arg.orientacion = "E";
var btnVAC3:botonFuenteVoltajeAC = new botonFuenteVoltajeAC(arg);

arg.columna = 3;
arg.orientacion = "W";
var btnVAC4:botonFuenteVoltajeAC = new botonFuenteVoltajeAC(arg);

// ---------------------------------
//            FILA: 3
// ---------------------------------
//    RESISTENCIAS Y CAPACITORES
// ---------------------------------
arg.fila = 3;
arg.color = amarillo;
// --- BOTONES >RESISTENCIAS: Argumentos variables ---
arg.columna = 0;
arg.orientacion = "W";
var btnRH:botonResistencia = new botonResistencia(arg);

arg.columna = 1;
arg.orientacion = "N";
var btnRV:botonResistencia = new botonResistencia(arg);

// --- BOTONES >CAPACITORES: Argumentos variables ---
arg.columna = 2;
arg.color = amarillo;
arg.orientacion = "W";
var btnCH:botonCapacitor = new botonCapacitor(arg);

arg.columna = 3;
arg.orientacion = "N";
var btnCV:botonCapacitor = new botonCapacitor(arg);

// ---------------------------------
//            FILA: 4
// ---------------------------------
//            ALAMBRES
// ---------------------------------
arg.fila = 4;
arg.color = amarillo;
```

```
// --- BOTONES >ALAMBRES: Argumentos variables ---
arg.columna = 0;
arg.orientacion = "W";
var btnA1H:botonAlambreLargo = new botonAlambreLargo(arg);

arg.columna = 1;
arg.orientacion = "N";
var btnA1V:botonAlambreLargo = new botonAlambreLargo(arg);

arg.columna = 2;
arg.orientacion = "W";
var btnA2H:botonAlambreCorto = new botonAlambreCorto(arg);

arg.columna = 3;
arg.orientacion = "N";
var btnA2V:botonAlambreCorto = new botonAlambreCorto(arg);

// ---------------------------------
//            FILA: 5
// ---------------------------------
//                              TIERRA
// ---------------------------------
arg.fila = 5;
arg.color = amarillo;
// --- BOTON >CONEXION: Argumentos variables ---
/*arg.columna = 0;
arg.orientacion = "W";
var btnConexion:botonConexion = new botonConexion(arg);*/

arg.columna = 0;
arg.orientacion = "N";
var btnGround0:botonTierra = new botonTierra(arg);

arg.columna = 1;
arg.orientacion = "W";
var btnGround1:botonTierra = new botonTierra(arg);

arg.columna = 2;
arg.orientacion = "S";
var btnGround2:botonTierra = new botonTierra(arg);

arg.columna = 3;
arg.orientacion = "E";
var btnGround3:botonTierra = new botonTierra(arg);

// ---------------------------------
//            FILA: 6
// ---------------------------------
//  AMPLIFICADOR OPERACIONAL IDEAL
// ---------------------------------
arg.fila = 6;
arg.color = verde;
// --- BOTONES >OpAmp: Argumentos variables ---
arg.columna = 0;
arg.orientacion = "E";
// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <W> y <S> ---
```

```
                var btnOpAmp1:botonOpAmp = new botonOpAmp(arg);
        }

        public function generaPosicionesBotones():void {

                // ---------------------------------------------------------
                // Estructura de DATOS:
                // posicionBoton[fila][columna] = Point(coord_x, coord_y]
                // 8 filas x 6 columnas
                // ---------------------------------------------------------
                posicionBoton = new Array();

                for (var iy=0; iy<=7; iy++) {
                        posicionBoton[iy] = new Array();
                        for (var ix=0; ix<=5; ix++) {
                                posicionBoton[iy][ix] = new Point(posicionInicial+ix*

    (tamanioBoton+separacion),

    posicionInicial+iy*

    (tamanioBoton+separacion));
                        }
                }
        }

        public function posicionaBoton(fila:uint, columna:uint):Point {

                // ---------------------------------------------------------
                //  Consulta coordenadas del boton en funcion de su "fila"
                //                                          y su "renglon" solicitado
                // ---------------------------------------------------------
                return posicionBoton[fila][columna];
        }

    }
}
```

```
package com.fabrica.consumidor.botonesComponentes.alambre
{
	// ************************************************************
	//
	//		Clase:	botonAlambreLargo (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.alambre.FabricarAlambre1;

	public class botonAlambreLargo {

		private var arg:Object;
		// ---------------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                          los botones
		// arg.spEditor                    --> "sprite" en donde se colocan
		//                                                          los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                          del boton
		// arg.color                       --> color del "marco" del boton
		// arg.orientacion                 --> orientacion del "logo" del boton
		//                                                          y del componente:
		//                                                          "W" <- | "N" ^ | "E" -> | "S"
v
		// ---------------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
		public function botonAlambreLargo(arg):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del componente ---
			var overLogo:mc_Alambre1 = new mc_Alambre1();
			var downLogo:mc_Alambre1 = new mc_Alambre1();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "W":                                                    // "W" <-
				break;
				case "N":
					overLogo.rotation = 90;          // "N" ^
					downLogo.rotation = 90;
				break;
				case "E":
					overLogo.rotation = 180;         // "E" ->
					downLogo.rotation = 180;
				break;
				case "S":
					overLogo.rotation = 270;         // "S" v
					downLogo.rotation = 270;
				break;
			}

			// --- Creacion del boton ---
			var btn:boton = new boton(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuComponentes.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {

			// ------------------------------------------------------
			//                          Fabricacion del componente
			// ------------------------------------------------------
			var componente:FabricarComponente = new FabricarAlambre1();
			componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
		}

	}
}
```

```
package com.fabrica.consumidor.botonesComponentes.alambre
{
	// ************************************************************
	//
	//		Clase:	botonAlambreCorto (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.alambre.FabricarAlambre2;

	public class botonAlambreCorto {

		private var arg:Object;
		// ------------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                        los botones
		// arg.spEditor                --> "sprite" en donde se colocan
		//                                                        los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                        del boton
		// arg.color                   --> color del "marco" del boton
		// arg.orientacion             --> orientacion del "logo" del boton
		//                                                        y del componente:
		//                                                        "W" <- | "N" ^ | "E" -> | "S"
v
		// ------------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
		public function botonAlambreCorto(arg):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del componente ---
			var overLogo:mc_Alambre2 = new mc_Alambre2();
			var downLogo:mc_Alambre2 = new mc_Alambre2();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "W":                                                    // "W" <-
				break;
				case "N":
					overLogo.rotation = 90;          // "N" ^
					downLogo.rotation = 90;
				break;
				case "E":
					overLogo.rotation = 180;         // "E" ->
					downLogo.rotation = 180;
				break;
				case "S":
					overLogo.rotation = 270;         // "S" v
					downLogo.rotation = 270;
				break;
			}

			// --- Creacion del boton ---
			var btn:boton = new boton(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuComponentes.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {

			// ------------------------------------------------------
			//                        Fabricacion del componente
			// ------------------------------------------------------
			var componente:FabricarComponente = new FabricarAlambre2();
			componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
		}

	}
}
```

```
package com.fabrica.consumidor.botonesComponentes.capacitor
{
	// ************************************************************
	//
	//		Clase:	botonCapacitor (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.capacitor.FabricarCapacitor;

	public class botonCapacitor{

		private var arg:Object;
		// ---------------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                          los botones
		// arg.spEditor                    --> "sprite" en donde se colocan
		//                                                          los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                          del boton
		// arg.color                       --> color del "marco" del boton
		// arg.orientacion                 --> orientacion del "logo" del boton
		//                                                          y del componente:
		//                                                          "W" <- | "N" ^ | "E" -> | "S"
v
		// ---------------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
		public function botonCapacitor(arg:Object):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del componente ---
			var overLogo:mc_Capacitor = new mc_Capacitor();
			var downLogo:mc_Capacitor = new mc_Capacitor();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "W":                                                    // "W" <-
				break;
				case "N":
					overLogo.rotation = 90;          // "N" ^
					downLogo.rotation = 90;
				break;
				case "E":
					overLogo.rotation = 180;         // "E" ->
					downLogo.rotation = 180;
				break;
				case "S":
					overLogo.rotation = 270;         // "S" v
					downLogo.rotation = 270;
				break;
			}

			// --- Creacion del boton ---
			var btn:boton = new boton(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuComponentes.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {

			// ------------------------------------------------------
			//                          Fabricacion del componente
			// ------------------------------------------------------
			var componente:FabricarComponente = new FabricarCapacitor();
			componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
		}

	}
}
```

```
package com.fabrica.consumidor.botonesComponentes.resistencia
{
	// ************************************************************
	//
	//		Clase:	botonResistencia (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.resistencia.FabricarResistencia;

	public class botonResistencia {

		private var arg:Object;
		// ---------------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                          los botones
		// arg.spEditor                    --> "sprite" en donde se colocan
		//                                                          los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                          del boton
		// arg.color                       --> color del "marco" del boton
		// arg.orientacion                 --> orientacion del "logo" del boton
		//                                                          y del componente:
		//                                                          "W" <- | "N" ^ | "E" -> | "S"
v
		// ---------------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
		public function botonResistencia(arg:Object):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del componente ---
			var overLogo:mc_Resistencia = new mc_Resistencia();
			var downLogo:mc_Resistencia = new mc_Resistencia();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "W":                                                    // "W" <-
				break;
				case "N":
					overLogo.rotation = 90;          // "N" ^
					downLogo.rotation = 90;
				break;
				case "E":
					overLogo.rotation = 180;         // "E" ->
					downLogo.rotation = 180;
				break;
				case "S":
					overLogo.rotation = 270;         // "S" v
					downLogo.rotation = 270;
				break;
			}

			// --- Creacion del boton ---
			var btn:boton = new boton(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuComponentes.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {

			// ------------------------------------------------------
			//                        Fabricacion del componente
			// ------------------------------------------------------
			var componente:FabricarComponente = new FabricarResistencia();
			componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
		}

	}
}
```

```
package com.fabrica.consumidor.botonesComponentes.fuenteVoltajeDC
{
	// ************************************************************
	//
	//		Clase:	botonFuenteVoltajeDC (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.fuentesVoltaje.FabricarVDC;

	public class botonFuenteVoltajeDC {

		private var arg:Object;
		// ----------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                        los botones
		// arg.spEditor                --> "sprite" en donde se colocan
		//                                                        los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                        del boton
		// arg.color                   --> color del "marco" del boton
		// arg.orientacion             --> orientacion del "logo" del boton
		//                                                        y del componente:
		//                                                        "W" <- | "N" ^ | "E" -> | "S"
v
		// ----------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
		public function botonFuenteVoltajeDC(arg):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del componente ---
			var overLogo:mc_VoltajeDC = new mc_VoltajeDC();
			var downLogo:mc_VoltajeDC = new mc_VoltajeDC();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "W":                                                    // "W" <-
				break;
				case "N":
					overLogo.rotation = 90;          // "N" ^
					downLogo.rotation = 90;
				break;
				case "E":
					overLogo.rotation = 180;         // "E" ->
					downLogo.rotation = 180;
				break;
				case "S":
					overLogo.rotation = 270;         // "S" v
					downLogo.rotation = 270;
				break;
			}

			// --- Creacion del boton ---
			var btn:boton = new boton(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuComponentes.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {

			// ------------------------------------------------------
			//                        Fabricacion del componente
			// ------------------------------------------------------
			var componente:FabricarComponente = new FabricarVDC();
			componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
		}

	}
}
```

```
package com.fabrica.consumidor.botonesComponentes.fuenteVoltajeAC
{
	// ************************************************************
	//
	//		Clase:	botonFuenteVoltajeAC (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.fuentesVoltaje.FabricarVAC;

	public class botonFuenteVoltajeAC {

		private var arg:Object;
		// ----------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                        los botones
		// arg.spEditor                --> "sprite" en donde se colocan
		//                                                        los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                        del boton
		// arg.color                   --> color del "marco" del boton
		// arg.orientacion             --> orientacion del "logo" del boton
		//                                                        y del componente:
		//                                                        "W" <- | "N" ^ | "E" -> | "S"
v
		// ----------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
		public function botonFuenteVoltajeAC(arg):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del componente ---
			var overLogo:mc_VoltajeAC = new mc_VoltajeAC();
			var downLogo:mc_VoltajeAC = new mc_VoltajeAC();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "W":                                                    // "W" <-
				break;
				case "N":
					overLogo.rotation = 90;          // "N" ^
					downLogo.rotation = 90;
				break;
				case "E":
					overLogo.rotation = 180;         // "E" ->
					downLogo.rotation = 180;
				break;
				case "S":
					overLogo.rotation = 270;         // "S" v
					downLogo.rotation = 270;
				break;
			}

			// --- Creacion del boton ---
			var btn:boton = new boton(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuComponentes.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {

			// ------------------------------------------------------
			//                        Fabricacion del componente
			// ------------------------------------------------------
			var componente:FabricarComponente = new FabricarVAC();
			componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
		}

	}
}
```

```
package com.fabrica.consumidor.botonesComponentes.fuenteCorrienteDC
{
	// ************************************************************
	//
	//		Clase:	botonFuenteCorrienteDC (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.fuentesCorriente.FabricarIDC;

	public class botonFuenteCorrienteDC {

		private var arg:Object;
		// ----------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                        los botones
		// arg.spEditor                --> "sprite" en donde se colocan
		//                                                        los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                        del boton
		// arg.color                   --> color del "marco" del boton
		// arg.orientacion             --> orientacion del "logo" del boton
		//                                                        y del componente:
		//                                                        "W" <- | "N" ^ | "E" -> | "S"
v
		// ----------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
        public function botonFuenteCorrienteDC(arg):void {

            // --- Recepcion de datos ---
            this.arg = arg;

            this.fila = arg.fila;
            this.columna = arg.columna;
            this.color = arg.color;
            this.orientacion = arg.orientacion;

            // --- Logos del componente ---
            var overLogo:mc_CorrienteDC = new mc_CorrienteDC();
            var downLogo:mc_CorrienteDC = new mc_CorrienteDC();

            // --- Seleccion de la orientacion del componente ---
            switch(arg.orientacion) {
                case "W":                                                   // "W" <-
                break;
                case "N":
                    overLogo.rotation = 90;         // "N" ^
                    downLogo.rotation = 90;
                break;
                case "E":
                    overLogo.rotation = 180;        // "E" ->
                    downLogo.rotation = 180;
                break;
                case "S":
                    overLogo.rotation = 270;        // "S" v
                    downLogo.rotation = 270;
                break;
            }

            // --- Creacion del boton ---
            var btn:boton = new boton(color);
            btn.agregarLogo(overLogo, downLogo);
            arg.spMenuComponentes.addChild(btn);
            btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
            btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
            btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
        }

        public function respuestaBoton(event:MouseEvent):void {

            // ------------------------------------------------------
            //                      Fabricacion del componente
            // ------------------------------------------------------
            var componente:FabricarComponente = new FabricarIDC();
            componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
        }

    }
}
```

```
package com.fabrica.consumidor.botonesComponentes.amplificador_operacional
{
	// ************************************************************
	//
	//		Clase:	botonOpAmp (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.opamp.FabricarOpAmp;

	public class botonOpAmp {

		private var arg:Object;
		// ---------------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                          los botones
		// arg.spEditor                    --> "sprite" en donde se colocan
		//                                                          los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                          del boton
		// arg.color                       --> color del "marco" del boton
		// arg.orientacion                 --> orientacion del "logo" del boton
		//                                                          y del componente:
		//                                                          "W" <- | "N" ^ | "E" -> | "S"
v
		// ---------------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
		public function botonOpAmp(arg):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del componente ---
			var overLogo:mc_OpAmp = new mc_OpAmp();
			var downLogo:mc_OpAmp = new mc_OpAmp();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "E":                                                    // "E" ->
				break;
				case "W":                                                    // "W" <-
				break;
				case "S":
					overLogo.rotation = 270;          // "S" v
					downLogo.rotation = 270;
				break;
				default:
				trace(" ");
				trace("************************************************************");
				trace("*                        ERROR:  ORIENTACION NO VALIDA
*");
				trace("*   SOLO SE ACEPTAN LAS ORIENTACIONES: <W> y <S>
*");
				trace("************************************************************");
			}

			// --- Creacion del boton ---
			var btn:boton = new boton(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuComponentes.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {

			// ------------------------------------------------------------
			//                          Fabricacion del componente
			// ------------------------------------------------------------
			var componente:FabricarComponente = new FabricarOpAmp();
			componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
		}

	}
}
```

```
package com.fabrica.consumidor.botonesComponentes.tierra
{
	// ************************************************************
	//
	//		Clase:	botonTierra (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton
	import com.bin.boton;

	// Conexion con la fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;
	import com.fabrica.productor.componentes.conexion.FabricarTierra;

	public class botonTierra {

		private var arg:Object;
		// ----------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                    los botones
		// arg.spEditor                --> "sprite" en donde se colocan
		//                                                    los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                    del boton
		// arg.color                   --> color del "marco" del boton
		// arg.orientacion             --> orientacion del "logo" del boton
		//                                                    y del componente:
		//                                                    "W" <- | "N" ^ | "E" -> | "S"
v
		// ----------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;
```

```
		public function botonTierra(arg:Object):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del componente ---
			var overLogo:mc_Ground = new mc_Ground();
			var downLogo:mc_Ground = new mc_Ground();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "N":                                                    // "N" ^
				break;
				case "W":
					overLogo.rotation = 90;          // "W" <-
					downLogo.rotation = 90;
				break;
				case "S":
					overLogo.rotation = 180;         // "S" v
					downLogo.rotation = 180;
				break;
				case "E":
					overLogo.rotation = 270;         // "E" ->
					downLogo.rotation = 270;
				break;
			}

			// --- Creacion del boton ---
			var btn:boton = new boton(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuComponentes.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {

			// ------------------------------------------------------
			//                        Fabricacion del componente
			// ------------------------------------------------------
			var componente:FabricarComponente = new FabricarTierra();
			componente.Fabricar(arg.spEditor, arg.parametros, orientacion);
		}

	}
}
```

```
package com.fabrica.consumidor.registroBotones
{
	// ****************************************************************
	//
	//		Clase:	CrearBotonesCircuitos (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                                  mayo / 2010
	// ****************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Botones de componentes
	// Circuitos DC
	import com.fabrica.consumidor.botonesCircuitos.dc.botonDivisor1;
	// Circuitos AC
	import com.fabrica.consumidor.botonesCircuitos.ac.botonFiltroLPActivo1;

	/*import com.fabrica.consumidor.botonesCircuitos.dc.botonRed1;
	import com.fabrica.consumidor.botonesCircuitos.dc.botonRed2;
	import com.fabrica.consumidor.botonesCircuitos.dc.botonInversor;
	import com.fabrica.consumidor.botonesCircuitos.dc.botonNoInversor;
	import com.fabrica.consumidor.botonesCircuitos.dc.botonDiferencial;

	import com.fabrica.consumidor.botonesCircuitos.ac.botonLPRC;
	import com.fabrica.consumidor.botonesCircuitos.ac.botonLPActivo;
	import com.fabrica.consumidor.botonesCircuitos.ac.botonHPActivo;
	import com.fabrica.consumidor.botonesCircuitos.ac.botonAllPassActivo;
	import com.fabrica.consumidor.botonesCircuitos.ac.botonStateVar;*/

	public class CrearBotonesCircuitos extends Sprite {

		// ---------------------------------
		//                                  COLORES
		// ---------------------------------
		public var verde:Number = 0x00ff99;
		public var naranja:Number = 0xff6633;
		public var naranja2:Number = 0xff9900;
		public var azul:Number = 0x0099ff;
		public var azul2:Number = 0x0066ff;
		public var amarillo:Number = 0xffff00;
		public var violeta:Number = 0xff00ff;
		public var rojo:Number = 0xff0000;
		public var rojo2:Number = 0xcc0000;
		public var cafe:Number = 0xcc6633;

		public var blanco:Number = 0xffffff;
		public var gris:Number = 0x666666;
		public var negro:Number = 0x000000;

		// Posicion de botones
		public var posicionBoton:Array;
		public var posicionInicial:uint = 10;
		public var tamanioBoton:uint = 150;
		public var separacion:uint = 2;

		// --- Datos ---
		public var spMenuCircuitos:Sprite;
		public var spEditor:Sprite;
		public var parametros:Object

		public function CrearBotonesCircuitos(spMenuCircuitos:Sprite,
```

```
                                                                    spEditor:Sprite,
                                                                    parametros:Object):void {

            // --- Recepcion de datos ---
            this.spMenuCircuitos = spMenuCircuitos;
            this.spEditor = spEditor;
            this.parametros = parametros;

            // --- Genera posisiones de Botones en 8 filas y 6 columnas ---
            generaPosicionesBotones();
            // --- Crear botones de circuitos ---
            generarBotonesCircuitos();
}

public function generarBotonesCircuitos():void {

            // ------------------------------------------------------------
            // Estructura de Datos del argumento "arg" para solicitar
            // la creacion de un boton:
            //
            // +++ Constantes +++
            // arg.spMenuComponentes --> "sprite" en donde se colocan
            //                                                                    los botones
            // arg.spEditor                          --> "sprite" en donde se colocan
            //                                                                    los componentes
            // arg.generaCoordenadas --> "objeto" parametros
            // +++ Variables +++
            // arg.fila y arg. columna > coordenadas para colocacion
            //                                                                    del boton
            // arg.color                             --> color del "marco" del boton
            // arg.orientacion               --> orientacion del "logo" del boton
            //                                                                    y del componente:
            //                                                  "W" <- | "N" ^ | "E" -> | "S" v
            // ------------------------------------------------------------
            var arg:Object = new Object();

            // --- Argumentos constantes ---
            arg.spMenuCircuitos = spMenuCircuitos;
            arg.spEditor = spEditor;
            arg.generaCoordenadas = this;
            arg.parametros = parametros;

            // ----------------------------------
            //          FILA: 0
            // ----------------------------------
            //                      CIRCUITOS DC
            // ----------------------------------
            arg.fila = 0;
            arg.color = azul;
            // --- BOTONES > CIRCUITOS DC: Argumentos variables ---
            arg.columna = 0;
            arg.orientacion = "W";
            var btnDivisor1:botonDivisor1 = new botonDivisor1(arg);

            arg.columna = 1;
            arg.orientacion = "W";

            var btnFiltroLPActivo1:botonFiltroLPActivo1 = new botonFiltroLPActivo1(arg);

            /*arg.columna = 2;
            arg.orientacion = "E";
            var btnVDC3:botonFuenteVoltajeDC = new botonFuenteVoltajeDC(arg);

            arg.columna = 3;
            arg.orientacion = "W";
            var btnVDC4:botonFuenteVoltajeDC = new botonFuenteVoltajeDC(arg);*/

            // ----------------------------------
            //          FILA: 1
            // ----------------------------------
```

```
//    FUENTES DE CORRIENTE DC
// ----------------------------------
/*arg.fila = 1;
arg.color = azul;
// --- BOTONES >FUENTE DE VOLTAJE DC: Argumentos variables ---
arg.columna = 0;
arg.orientacion = "N";
var btnIDC1:botonFuenteCorrienteDC = new botonFuenteCorrienteDC(arg);

arg.columna = 1;
arg.orientacion = "S";
var btnIDC2:botonFuenteCorrienteDC = new botonFuenteCorrienteDC(arg);

arg.columna = 2;
arg.orientacion = "E";
var btnIDC3:botonFuenteCorrienteDC = new botonFuenteCorrienteDC(arg);

arg.columna = 3;
arg.orientacion = "W";
var btnIDC4:botonFuenteCorrienteDC = new botonFuenteCorrienteDC(arg);*/

// ----------------------------------
//            FILA: 2
// ----------------------------------
//    FUENTES DE VOLTAJE AC
// ----------------------------------
/*arg.fila = 2;
arg.color = azul;
// --- BOTONES >FUENTE DE VOLTAJE AC: Argumentos variables ---
arg.columna = 0;
arg.color = azul;
arg.orientacion = "N";
var btnVAC1:botonFuenteVoltajeAC = new botonFuenteVoltajeAC(arg);

arg.columna = 1;
arg.orientacion = "S";
var btnVAC2:botonFuenteVoltajeAC = new botonFuenteVoltajeAC(arg);

arg.columna = 2;
arg.orientacion = "E";
var btnVAC3:botonFuenteVoltajeAC = new botonFuenteVoltajeAC(arg);

arg.columna = 3;
arg.orientacion = "W";
var btnVAC4:botonFuenteVoltajeAC = new botonFuenteVoltajeAC(arg);*/

// ----------------------------------
//            FILA: 3
// ----------------------------------
//     RESISTENCIAS Y CAPACITORES
// ----------------------------------
/*arg.fila = 3;
arg.color = amarillo;
// --- BOTONES >RESISTENCIAS: Argumentos variables ---
arg.columna = 0;
arg.orientacion = "W";
var btnRH:botonResistencia = new botonResistencia(arg);

arg.columna = 1;
arg.orientacion = "N";
var btnRV:botonResistencia = new botonResistencia(arg);

// --- BOTONES >CAPACITORES: Argumentos variables ---
arg.columna = 2;
arg.color = amarillo;
arg.orientacion = "W";
var btnCH:botonCapacitor = new botonCapacitor(arg);

arg.columna = 3;
arg.orientacion = "N";
```

```
var btnCV:botonCapacitor = new botonCapacitor(arg);*/

// ---------------------------------
//            FILA: 4
// ---------------------------------
//            ALAMBRES
// ---------------------------------
/*arg.fila = 4;
arg.color = amarillo;
// --- BOTONES >ALAMBRES: Argumentos variables ---
arg.columna = 0;
arg.orientacion = "W";
var btnA1H:botonAlambreLargo = new botonAlambreLargo(arg);

arg.columna = 1;
arg.orientacion = "N";
var btnA1V:botonAlambreLargo = new botonAlambreLargo(arg);

arg.columna = 2;
arg.orientacion = "W";
var btnA2H:botonAlambreCorto = new botonAlambreCorto(arg);

arg.columna = 3;
arg.orientacion = "N";
var btnA2V:botonAlambreCorto = new botonAlambreCorto(arg);*/

// ---------------------------------
//            FILA: 5
// ---------------------------------
//                                    TIERRA
// ---------------------------------
/*arg.fila = 5;
arg.color = amarillo;*/
// --- BOTON >CONEXION: Argumentos variables ---
/*arg.columna = 0;
arg.orientacion = "W";
var btnConexion:botonConexion = new botonConexion(arg);*/

/*arg.columna = 0;
arg.orientacion = "N";
var btnGround0:botonTierra = new botonTierra(arg);

arg.columna = 1;
arg.orientacion = "W";
var btnGround1:botonTierra = new botonTierra(arg);

arg.columna = 2;
arg.orientacion = "S";
var btnGround2:botonTierra = new botonTierra(arg);

arg.columna = 3;
arg.orientacion = "E";
var btnGround3:botonTierra = new botonTierra(arg);*/

// ---------------------------------
//            FILA: 6
// ---------------------------------
//  AMPLIFICADOR OPERACIONAL IDEAL
// ---------------------------------
/*arg.fila = 6;
arg.color = verde;
// --- BOTONES >OpAmp: Argumentos variables ---
arg.columna = 0;
arg.orientacion = "E";
// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <W> y <S> ---
var btnOpAmp1:botonOpAmp = new botonOpAmp(arg);*/


// ---------------------------------
//            FILA: 5
```

```
		// ---------------------------------
		//		TRANSISTORES NPN
		// ---------------------------------
		/*arg.fila = 4;
		arg.color = verde;
		// --- BOTONES >TRANSISTORES NPN: Argumentos variables ---
		arg.columna = 0;
		arg.orientacion = "W";
		// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <W> y <S> ---
		var btnNPN1:botonTransistorNPN = new botonTransistorNPN(arg);

		arg.columna = 1;
		arg.orientacion = "S";
		// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <W> y <S> ---
		var btnNPN2:botonTransistorNPN = new botonTransistorNPN(arg);

		arg.columna = 2;
		arg.orientacion = "E";
		// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <E> y <S> ---
		var btnNPNFlip1:botonTransistorNPNF = new botonTransistorNPNF(arg);

		arg.columna = 3;
		arg.orientacion = "S";
		// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <E> y <S> ---
		var btnNPNFlip2:botonTransistorNPNF = new botonTransistorNPNF(arg);*/

		// ---------------------------------
		//		FILA: 5
		// ---------------------------------
		//		TRANSISTORES PNP
		// ---------------------------------
		/*arg.fila = 5;
		arg.color = verde;
		// --- BOTONES >TRANSISTORES NPN: Argumentos variables ---
		arg.columna = 0;
		arg.orientacion = "W";
		// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <W> y <S> ---
		var btnPNP1:botonTransistorPNP = new botonTransistorPNP(arg);

		arg.columna = 1;
		arg.orientacion = "S";
		// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <W> y <S> ---
		var btnPNP2:botonTransistorPNP = new botonTransistorPNP(arg);

		arg.columna = 2;
		arg.orientacion = "E";
		// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <E> y <S> ---
		var btnPNPFlip1:botonTransistorPNPF = new botonTransistorPNPF(arg);

		arg.columna = 3;
		arg.orientacion = "S";
		// --- SOLO SE ACEPTAN LAS ORIENTACIONES: <E> y <S> ---
		var btnPNPFlip2:botonTransistorPNPF = new botonTransistorPNPF(arg);*/

}

public function generaPosicionesBotones():void {

		// ------------------------------------------------------------
		// Estructura de DATOS:
		// posicionBoton[fila][columna] = Point(coord_x, coord_y]
		// 3 filas x 6 columnas
		// ------------------------------------------------------------
		posicionBoton = new Array();

		for (var iy=0; iy<2; iy++) {
			posicionBoton[iy] = new Array();
			for (var ix=0; ix<=5; ix++) {
				posicionBoton[iy][ix] = new Point(posicionInicial+ix*
```

```
                                                        (tamanioBoton+separacion),
                                                posicionInicial+iy*
                                                        (tamanioBoton+separacion));
                                        }
                                }
                        }

                        public function posicionaBoton(fila:uint, columna:uint):Point {

                                // ------------------------------------------------------------
                                //  Consulta coordenadas del boton en funcion de su "fila"
                                //                                                  y su "renglon" solicitado
                                // ------------------------------------------------------------
                                return posicionBoton[fila][columna];
                        }

                }
}


package com.fabrica.consumidor.botonesCircuitos.dc
{
        // ************************************************************
        //
        //      Clase:  botonDivisor1 (CLIENTE)
        //
        //      Patron de Programacion:
        //
        //              "Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
        //
        //                                                              mayo / 2010
        // ************************************************************

        // Flash principal
        import flash.display.*;
        // Eventos
        import flash.events.*;
        // Matematicas
        import flash.geom.Point;

        // Crear un boton de circuito
        import com.bin.botonCircuito;

        // Conexion con la fabrica
        import com.fabrica.productor.circuitos.manejadorCircuito;

        public class botonDivisor1 {

                private var arg:Object;
                // ------------------------------------------------------------
                // Estructura de Datos del argumento "arg" para solicitar
                // la creacion de un boton:
                //
                // +++ Constantes +++
                // arg.spMenuComponentes --> "sprite" en donde se colocan
                //                                                  los botones
                // arg.spEditor                 --> "sprite" en donde se colocan
                //                                                  los componentes
                // arg.generaCoordenadas --> "objeto" parametros
                // +++ Variables +++
```

```
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                    del boton
		// arg.color                    --> color del "marco" del boton
		// arg.orientacion              --> orientacion del "logo" del boton
		//                                                    y del componente:
		//                                                    "W" <- | "N" ^ | "E" -> | "S"
v
		// ----------------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;

		public function botonDivisor1(arg:Object):void {

			// --- Recepcion de datos ---
			this.arg = arg;

			this.fila = arg.fila;
			this.columna = arg.columna;
			this.color = arg.color;
			this.orientacion = arg.orientacion;

			// --- Logos del circuito ---
			var overLogo:mc_Circuito_Divisor01 = new mc_Circuito_Divisor01();
			var downLogo:mc_Circuito_Divisor01 = new mc_Circuito_Divisor01();

			// --- Seleccion de la orientacion del componente ---
			switch(arg.orientacion) {
				case "W":                                                    // "W" <-
				break;
				case "N":
					overLogo.rotation = 90;          // "N" ^
					downLogo.rotation = 90;
				break;
				case "E":
					overLogo.rotation = 180;         // "E" ->
					downLogo.rotation = 180;
				break;
				case "S":
					overLogo.rotation = 270;         // "S" v
					downLogo.rotation = 270;
				break;
			}

			// --- Creacion del boton ---
			var btn:botonCircuito = new botonCircuito(color);
			btn.agregarLogo(overLogo, downLogo);
			arg.spMenuCircuitos.addChild(btn);
			btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
			btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
			btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
		}

		public function respuestaBoton(event:MouseEvent):void {
```

```
                    // ----------------------------------------------------------
                    //                         Fabricacion del circuito
                    // ----------------------------------------------------------
                    var circuito:manejadorCircuito = new manejadorCircuito("circuitoDivisor1",

                            arg.spEditor,

                            arg.parametros);
                }

        }
}
```

```
package com.fabrica.consumidor.botonesCircuitos.ac
{
	// ************************************************************
	//
	//		Clase:   botonFiltroLPActivo1 (CLIENTE)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Crear un boton de circuito
	import com.bin.botonCircuito;

	// Conexion con la fabrica
	import com.fabrica.productor.circuitos.manejadorCircuito;

	public class botonFiltroLPActivo1 {

		private var arg:Object;
		// ---------------------------------------------------------
		// Estructura de Datos del argumento "arg" para solicitar
		// la creacion de un boton:
		//
		// +++ Constantes +++
		// arg.spMenuComponentes --> "sprite" en donde se colocan
		//                                                          los botones
		// arg.spEditor                  --> "sprite" en donde se colocan
		//                                                          los componentes
		// arg.generaCoordenadas --> "objeto" parametros
		// +++ Variables +++
		// arg.fila y arg. columna > coordenadas para colocacion
		//                                                          del boton
		// arg.color                     --> color del "marco" del boton
		// arg.orientacion               --> orientacion del "logo" del boton
		//                                                          y del componente:
		//                                                          "W" <- | "N" ^ | "E" -> | "S"
v
		// ---------------------------------------------------------

		private var fila:uint;
		private var columna:uint;
		private var color:Number;
		private var orientacion:String;

		public function botonFiltroLPActivo1(arg:Object):void {
```

```
                // --- Recepcion de datos ---
                this.arg = arg;

                this.fila = arg.fila;
                this.columna = arg.columna;
                this.color = arg.color;
                this.orientacion = arg.orientacion;

                // --- Logos del circuito ---
                var overLogo:mc_FiltroActivoLP01 = new mc_FiltroActivoLP01();
                var downLogo:mc_FiltroActivoLP01 = new mc_FiltroActivoLP01();

                // --- Seleccion de la orientacion del componente ---
                switch(arg.orientacion) {
                        case "W":                                                       // "W" <-
                        break;
                        case "N":
                                overLogo.rotation = 90;         // "N" ^
                                downLogo.rotation = 90;
                        break;
                        case "E":
                                overLogo.rotation = 180;        // "E" ->
                                downLogo.rotation = 180;
                        break;
                        case "S":
                                overLogo.rotation = 270;        // "S" v
                                downLogo.rotation = 270;
                        break;
                }

                // --- Creacion del boton ---
                var btn:botonCircuito = new botonCircuito(color);
                btn.agregarLogo(overLogo, downLogo);
                arg.spMenuCircuitos.addChild(btn);
                btn.x = arg.generaCoordenadas.posicionaBoton(fila,columna).x;
                btn.y = arg.generaCoordenadas.posicionaBoton(fila,columna).y;
                btn.addEventListener(MouseEvent.CLICK, respuestaBoton);
        }

        public function respuestaBoton(event:MouseEvent):void {

                // ------------------------------------------------------------
                //                          Fabricacion del circuito
                // ------------------------------------------------------------
                var circuito:manejadorCircuito = new manejadorCircuito("filtroLPActivo1",

                        arg.spEditor,

                        arg.parametros);
        }

    }
}
```

```
package com.fabrica.productor.fabricacion
{
	// ****************************************************************
	//
	//		Clase ABSTRACTA:		Componente (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//				"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//																	mayo / 2010
	// ****************************************************************
	//
	// Esta clase constituye la definicion de la clase base para
	// establecer la FUNCIONALIDAD de todos los componentes utilizados
	// en el simulador
	//
	//		Clases derivadas:
	//
	//					- Componente1T			--> tierra
	//					- Componente2T			--> resistencia, capacitor,
	//										alambre largo (40
pixeles),
	//										fuentes de voltaje DC y
AC
	//										fuente de corriente de
DC
	//					- AlambreCorto2T   --> alambre corto (20 pixeles)
	//					- Componente3T			--> amplificador operacional
	//
	// ****************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;

	// Matematicas
	import flash.geom.*;

	// Error al intentar generar un objeto de esta clase abstracta
	import flash.errors.IllegalOperationError;

	// Animaciones
	import fl.transitions.Tween;
	import fl.transitions.TweenEvent;
	import fl.transitions.easing.*;

	// Ventana de edicion de componente
	import com.bin.ventanaEdicion;

	// Clase ABSTRACTA (debe ser "extendida"
	// y no intentar "generar objetos" de ella)
	public class Componente extends MovieClip {

		// ---------------------------------------------
		// Variables para MANEJO INTERNO del simulador
		// ---------------------------------------------
		// Contador de instancias de componentes
		static var i_numeroComponenteCreado:Number = 0;
		// Identificador del componente
		public var i_identificadorComponente:String;
		// Tipo de componente
		public var i_tipoComponente:String

		// ---------------------------------------------
		// Elementos Graficos del Componente
		// ---------------------------------------------
		public var contenedorGrafico:MovieClip;
		public var simboloGrafico:MovieClip;
```

```
public var lineaGrafica:MovieClip;
public var fondoGrafico:MovieClip;
public var corrienteGrafica:MovieClip;
public var potenciaGrafica:MovieClip;
// --- Animacion ---
private var lanzarComponenteGrafico:Tween;

// ----------------------------------------------
// Target para colocacion de componentes
// y dibujo del circuito en modo de Edicion
public var target:DisplayObjectContainer;
// ----------------------------------------------

// ----------------------------------------------
//              Variables para manejo del ususario
// ----------------------------------------------
public var u_identificadorComponente:String;
// Ventana de edicion de parametros del componente
public var ventanaEdicionPermitida:Boolean;
public var ventanaEdicionComponente:ventanaEdicion;
// ----------------------------------------------

// ----------------------------------------------
//               Arreglo para guardado de coordenadas
// ----------------------------------------------
public var datosComponente:Array;
// ----------------------------------------------
// Estructura de DATOS de "datosComponente":
// datosComponente[0] = componente(identificador,
//                                                                                                  numero_Punto_Rejilla,
xg, yg, zg)
// datosComponente[1] = terminal1(numero_Punto_Rejilla, xg, yg, zg)
// datosComponente[2] = terminal2(numero_Punto_Rejilla, xg, yg, zg)
// datosComponente[n] = terminaln(numero_Punto_Rejilla, xg, yg, zg)
// ----------------------------------------------

// ----------------------------------------------
//              PARAMETROS COMUNES COMPARTIDOS
// ----------------------------------------------
public var parametros:Object;
public var separacionEntrePuntosRejilla:Number;
public var numeroPuntosAnchoRejilla:Number;
public var numeroPuntosAltoRejilla:Number;
public var longitudComponente:Number;
// ----------------------------------------------
//                                                  Estructura de DATOS:
// puntosRejilla[numero_Punto_Rejilla] =
//      [fila, columna, x, y, z, xg, yg, zg]
//
//                        x, y, z    --> Coordenadas relativas
//                                                  a la esquina superior
//                                                  derecha del editor
//                        xg, yg, zg --> Coordenadas relativas
//                                                  al centro del area
//                                                  del editor
// ----------------------------------------------
public var puntosRejilla:Array;
public var numero_Punto_Rejilla:Number;
// ----------------------------------------------

// ----------------------------------------------
//              Variables auxiliares para ubicacion del
//                                                  componente
// ----------------------------------------------
public var numero_Fila_Rejilla_Componente:Number;
public var numero_Columna_Rejilla_Componente:Number;
public var numero_Punto_Inicial_Fila_Rejilla:Number;
public var numero_Punto_Inicial_Columna_Rejilla:Number;
// Offsets de colocacion del componente
public var offsetHorizontal:Number = 0;
```

```
                public var offsetVertical:Number = 0;
                // ------------------------------------------------

                // --- Constantes ---
                public var LN_LOG:Number = 0.434294;

                // ------------------------------------------------------------
                //
                //                                  Seccion de "inicializacion" de parametros
                //
                // ------------------------------------------------------------

                public function inicializaParametros(parametros:Object,

target:DisplayObjectContainer):void {

                        // ------------------------------------------
                        // Identificacion INTERNA del componente
                        // ------------------------------------------
                        i_identificadorComponente = i_tipoComponente +

        i_numeroComponenteCreado;
                        i_numeroComponenteCreado++;
                        u_identificadorComponente = i_identificadorComponente;
                        // ------------------------------------------
                        // Definicion del nombre de este "objeto" para
                        // referencia interna del simulador
                        // ------------------------------------------
                        this.name = i_identificadorComponente;
                        // ------------------------------------------

                        // ------------------------------------------
                        // Captura de parametros comunes
                        // ------------------------------------------
                        this.parametros = parametros;
                        separacionEntrePuntosRejilla =

parametros.separacionEntrePuntosRejilla;
                        numeroPuntosAnchoRejilla = parametros.numeroPuntosAnchoRejilla;
                        numeroPuntosAltoRejilla = parametros.numeroPuntosAltoRejilla;
                        longitudComponente = parametros.longitudComponente;
                        puntosRejilla = parametros.puntosRejilla;
                        // ------------------------------------------

                        // Define modo boton para este "objeto"
                        this.buttonMode = true;
                        // Inicializa capa de dibujo de edicion
                        this.target = target;

                        // Inicializa "listeners" para interactividad
                        inicializaListeners();

                        // Inicializa "datosComponente" para guardado de
                        // coordenadas
                        datosComponente = new Array();

                        // Inicializa ventana para edicion de parametros
                        // del componente
                        ventanaEdicionComponente = new ventanaEdicion(target,

                  this,parametros);
                }

                // ------------------------------------------------------------
                //
                //                                  Seccion de "edicion" para el componente
                //
                // ------------------------------------------------------------

                public function abrirPantallaEdicion(evt:MouseEvent):void {
```

```
                        // ------------------------------------------------------------
                        //              Metodo ABSTRACTO: se debe "definir" en una subclase
                        // ------------------------------------------------------------
                }

                // ------------------------------------------------------------
                //
                //                          Seccion de "listeners" para el componente
                //
                // ------------------------------------------------------------

                public function inicializaListeners():void {

                        // ------------------------------------------------------------
                        //  Inicializa "listeners" para:
                        //  - Pantalla de Edicion de parametros del componente
                        //          - Aplicacion de enfoque al componente
                        //          - Inciar movimiento de componente
                        //          - Eliminar componente
                        // ------------------------------------------------------------
                        simboloGrafico.doubleClickEnabled = true;
                        simboloGrafico.addEventListener(MouseEvent.DOUBLE_CLICK,

        abrirPantallaEdicion);
                        simboloGrafico.addEventListener(MouseEvent.CLICK,

        aplicarEnfoqueComponente);

                        this.addEventListener(MouseEvent.MOUSE_DOWN,

        iniciaMovimientoComponente);
                        this.addEventListener(KeyboardEvent.KEY_DOWN,

        eliminarComponente);
                }

                public function removerListeners():void {

                        // ------------------------------------------------------------
                        //                          Remover "listeners"
                        // ------------------------------------------------------------
                        simboloGrafico.removeEventListener(MouseEvent.DOUBLE_CLICK,

abrirPantallaEdicion);
                        simboloGrafico.removeEventListener(MouseEvent.CLICK,

aplicarEnfoqueComponente);
                        this.removeEventListener(MouseEvent.MOUSE_DOWN,

iniciaMovimientoComponente);

                        stage.removeEventListener(MouseEvent.MOUSE_MOVE,

        moverComponente);
                        stage.removeEventListener(MouseEvent.MOUSE_UP,

        detenerComponente);
                }

                // ------------------------------------------------------------
                //
                //                          Seccion para eliminar el componente
                //
                // ------------------------------------------------------------

                public function aplicarEnfoqueComponente(evt:MouseEvent=null):void {

                        // ------------------------------------------------------------
                        //                          Enfoque del componente
                        // ------------------------------------------------------------
```

```
                        stage.focus = this;
                }

                public function eliminarComponente(event:KeyboardEvent):void {

                        // ------------------------------------------------------------
                        //                         Eliminacion de componente
                        // ------------------------------------------------------------
                        var DELETE:Number = 46;
                        if (event.keyCode == DELETE) {
                                simboloGrafico.removeEventListener(MouseEvent.DOUBLE_CLICK,
abrirPantallaEdicion);
                                simboloGrafico.removeEventListener(MouseEvent.CLICK,
aplicarEnfoqueComponente);
                                this.removeEventListener(MouseEvent.MOUSE_DOWN,
iniciaMovimientoComponente);

                                stage.removeEventListener(MouseEvent.MOUSE_MOVE,
        moverComponente);
                                stage.removeEventListener(MouseEvent.MOUSE_UP,
        detenerComponente);
                                // --- Si la pantalla esta abierta, cerrarla ---
                                if (ventanaEdicionComponente.ventanaAbierta) {
                                        ventanaEdicionComponente.cerrar();
                                }

                                target.removeChild(this);
                        }
    }

                // ------------------------------------------------------------
                //
                //                         Seccion de "movimiento" del componente
                //
                // ------------------------------------------------------------

                public function iniciaMovimientoComponente(evt:MouseEvent):void {

                        contenedorGrafico = MovieClip(evt.currentTarget);
                        stage.addEventListener(MouseEvent.MOUSE_MOVE,moverComponente);
                        stage.addEventListener(MouseEvent.MOUSE_UP,detenerComponente);
                }

                public function moverComponente(e:MouseEvent):void {

                        contenedorGrafico.x = generaPosicionValidaX(target.mouseX);
                        contenedorGrafico.y = generaPosicionValidaY(target.mouseY);
                        // ------------------------------------------------------------
                        // Calculo del "numero de punto" de la rejilla en donde estan
                        // colocadas las terminales  del componente, considerando que
                        // el punto de registro del componente se encuentra al CENTRO
                        // de este, es decir, contenedorGrafico.x y contenedorGrafico.y
                        // son las coordenadas del CENTRO del componente.
                        // ------------------------------------------------------------
                        numero_Fila_Rejilla_Componente = contenedorGrafico.y/
separacionEntrePuntosRejilla+10;
                        numero_Columna_Rejilla_Componente = contenedorGrafico.x/
separacionEntrePuntosRejilla+10;
                        numero_Punto_Inicial_Fila_Rejilla = numero_Fila_Rejilla_Componente*
(numeroPuntosAnchoRejilla + 1);
                        numero_Punto_Inicial_Columna_Rejilla = numero_Columna_Rejilla_Componente;
```

```
                                numero_Punto_Rejilla = ((contenedorGrafico.x)/

separacionEntrePuntosRejilla + 10)+

numero_Punto_Inicial_Fila_Rejilla;
                                // ---------------------------------------------------------
                                // Guardado de los datos del componente en un arreglo de datos:
                                // "datosComponente"
                                // ---------------------------------------------------------

                                datosComponente[0] = new Array();
                                datosComponente[0].push(i_identificadorComponente);                                    //
[0][0]
                                datosComponente[0].push(numero_Punto_Rejilla);
            // [0][1]
                                datosComponente[0].push(puntosRejilla[numero_Punto_Rejilla].xg); // [0][2]
                                datosComponente[0].push(puntosRejilla[numero_Punto_Rejilla].yg); // [0][3]
                                datosComponente[0].push(0); // coordenada "z" del componente          // [0][4]
                                datosComponente[0].push(new Array()); // Ubicacion(t) entre nodos "z" [0][5]
                                datosComponente[0].push(new Array()); // Distancia(t) entre nodos       [0][6]
                                datosComponente[0].push(new Array()); // Inclinacion(t) entre nodos     [0][7]

                                actualiza_Coordenadas_Terminales();

                                // Actualiza "lineaGrafica" de conexion a la ventana de edicion
                                if (ventanaEdicionComponente.ventanaAbierta) {
                                        ventanaEdicionComponente.actualizaPosicion(this);
                                }

                                e.updateAfterEvent();
                        }

                        public function actualiza_Coordenadas_Terminales():void {
                                // ---------------------------------------------------------
                                //              Metodo ABSTRACTO: se debe "definir" en una subclase
                                // ---------------------------------------------------------
                        }

                        internal function generaPosicionValidaX(posicionMouseX:Number):Number {

                                // --- Limite derecho de la rejilla de dibujo ---
                                var limiteXMaximoPosicionRejilla:Number = separacionEntrePuntosRejilla*

            (numeroPuntosAnchoRejilla/2) - offsetHorizontal;
                                var limiteXMinimoPosicionRejilla:Number = -separacionEntrePuntosRejilla*

            (numeroPuntosAnchoRejilla/2) + offsetHorizontal;

                                if (posicionMouseX > limiteXMaximoPosicionRejilla) {
                                        return limiteXMaximoPosicionRejilla;
                                }
                                // --- Limite izquierdo de la rejilla de dibujo ---
                                if (posicionMouseX < limiteXMinimoPosicionRejilla) {
                                        return limiteXMinimoPosicionRejilla;
                                }
                                // --- Dentro de la rejilla de dibujo ---
                                return separacionEntrePuntosRejilla*Math.round(posicionMouseX/

                          separacionEntrePuntosRejilla);
                        }

                        internal function generaPosicionValidaY(posicionMouseY:Number):Number {

                                // --- Limite inferior de la rejilla de dibujo ---
                                var limiteYMaximoPosicionRejilla:Number = separacionEntrePuntosRejilla*

            (numeroPuntosAltoRejilla/2) - offsetVertical;
                                var limiteYMinimoPosicionRejilla:Number = -separacionEntrePuntosRejilla*

            (numeroPuntosAltoRejilla/2) + offsetVertical;
```

```
                        if (posicionMouseY > limiteYMaximoPosicionRejilla) {
                                return limiteYMaximoPosicionRejilla;
                        }
                        // --- Limite superior de la rejilla de dibujo ---
                        if (posicionMouseY < limiteYMinimoPosicionRejilla) {
                                return limiteYMinimoPosicionRejilla;
                        }
                        // --- Dentro de la rejilla de dibujo ---
                        return separacionEntrePuntosRejilla*Math.round(posicionMouseY/

separacionEntrePuntosRejilla);
                }

                internal function detenerComponente(e:MouseEvent):void {

                        contenedorGrafico.stopDrag();
                        stage.removeEventListener(MouseEvent.MOUSE_MOVE, moverComponente);
                }

                // ----------------------------------------------------------
                //
                //                      Seccion de colocacion inicial del componente
                //
                // ----------------------------------------------------------

                internal function colocacionInicial():void {

                        // --- Colocacion incial del componente cuando se genera
                        //     oprimir su boton de creacion correspondiente

                        //lanzarComponenteGrafico = new Tween(this, "x",
                        //Regular.easeOut, 300, parametros.posicionInicialComponenteX,
                        //0.2, true);
                        //lanzarComponenteGrafico = new Tween(this, "y",
                        //Regular.easeOut, 0, parametros.posicionInicialComponenteY,
                        //0.2, true);
                        //lanzarComponenteGrafico.addEventListener(TweenEvent.MOTION_FINISH,
                        //terminaTween);

                        //lanzarComponenteGrafico = new Tween(this, "scaleX",
                        //Regular.easeOut, 0, 1, 0.2, true);
                        //lanzarComponenteGrafico = new Tween(this, "scaleY",
                        //Regular.easeOut, 0, 1, 0.2, true);

                        this.x = parametros.posicionInicialComponenteX;
                        this.y = parametros.posicionInicialComponenteY;
                }

                public function terminaTween(evnt:Event):void {

                        setComponente();
                        lanzarComponenteGrafico.removeEventListener(TweenEvent.MOTION_FINISH,

                terminaTween);
                }

                public function setComponente():void {
                        //lanzarComponenteGrafico = new Tween(this, "x",
                        //Regular.easeOut, this.x, datosComponente[0][2], 0.2, true);
                        //lanzarComponenteGrafico = new Tween(this, "y",
                        //Regular.easeOut, this.y, datosComponente[0][3], 0.2, true);

                        this.x = datosComponente[0][2];
                        this.y = datosComponente[0][3];
                }

        }
}
```

```
package com.fabrica.productor.fabricacion
{
	// ************************************************************
	//
	//		Clase:	Componente1T (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "Componente"
	//
	// ************************************************************

	public class Componente1T extends Componente {

		// ---------------------------------------------
		//		Variables para manejo del ususario
		// ---------------------------------------------
		// Datos del componente para el susuario
		public var u_nombreComponente:String;
		public var u_valorComponente:Number;
		public var u_unidadesComponente:String;
		// ---------------------------------------------

		// ---------------------------------------------
		// Potenciales de las "terminales" del componente
		// ---------------------------------------------
		public var Vnodo1:Number; // Potencial de la
                                                                // "terminal1" (coord. "z")
		// ---------------------------------------------
		// ---------------------------------------------
		// Orientacion para colocacion del componente
		public var orientacion:String;
		// ---------------------------------------------

		public function especificaComponente(orientacion:String):void {

			u_identificadorComponente = i_identificadorComponente;
			this.orientacion = orientacion;

			// ------------------------------
			//  Orientacion del componente
			// ------------------------------
			switch (orientacion) {
				case "N":
					// ********************************
					//                          (1) o
					//                              |
					//                          (2) o
					// ********************************
					rotation = 0;
```

```
                        break;

                        case "W":
                        // ********************************
                                //
                                //                  (1) o -- o (2)
                                //
                                // ********************************

                                rotation = 90;
                        break;

                        case "S":
                                // ********************************
                                //                              (2) o
                                //                                  |
                                //                              (1) o
                                // ********************************

                                rotation = 180;
                        break;

                        case "E":
                                // ********************************
                                //
                                //                  (2) o -- o (1)
                                //
                                // ********************************
                                rotation = 270;
                        break;

                }
        }

        override public function actualiza_Coordenadas_Terminales():void {

                // ----------------------------------------------------------
                // Guardado de los datos del componente en un arreglo de datos:
                // "datosComponente"
                // ----------------------------------------------------------
                datosComponente[1] = new Array();
                // Asignacion de coordenadas a la terminal1 del Componente
                // Se asignan las mismas coordenadas del componente a la "terminal1"
                datosComponente[1][0] = datosComponente[0][1];          //
numero_Punto_Rejilla
                datosComponente[1][1] = datosComponente[0][2];          // coordenada "x"
                datosComponente[1][2] = datosComponente[0][3];          // coordenada "y"
                datosComponente[1][3] = datosComponente[0][4];          // coordenada "z"
                // --- V1(t) (comportamiento en AC) ---
                datosComponente[1].push(new Array());           // V1(t) (4)

                // ----------------------------------------------------------
                //            Para fines de pruebas de funcionamiento del simulador
                // ----------------------------------------------------------
                //trace("Componente1T:: datosComponente[0]= "+datosComponente[0]);
                //trace("Componente1T:: datosComponente[1]= "+datosComponente[1]);
```

```
		}

		// ----------------------------------------------------------
		//
		//                                   Seccion de SIMULACION EN AC
		//
		// ----------------------------------------------------------

		public function activarComportamiento3DComplejo():void {
			// --- Posicion "z" del contenedor ---
			var contenedor_z:Number;

			removerListeners();

			// --- Ubicacion(t) entre nodos "z": [0][5] ---
			datosComponente[0][5] = new Array();

			// Asignacion del "arreglo de voltajes nodales"
			// a las terminales del componente
			V1_t = datosComponente[1][4];

			for (i in V1_t) {
				contenedor_z = V1_t[i];
				datosComponente[0][5].push(contenedor_z);
			}
		}

		// ----------------------------------------------------------
		//
		//                   Seccion "onEnterFrame" llamada desde el motor3D_ac
		//
		// ----------------------------------------------------------

		public function asignaPotencial(t:Number):void {

			// Rota, escala y orienta el componente
			contenedor_z = datosComponente[0][5][t];
			this.z = contenedor_z;
		}

		public function desactivarComportamiento3DComplejo():void {

			this.z = 0;
			inicializaListeners();
		}

	}
}
```

```
package com.fabrica.productor.fabricacion
{
	// ****************************************************************
	//
	//		Clase:	Componente2T (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                                  mayo / 2010
	// ****************************************************************
	//
	//		Clase base heredada: "Componente"
	//
	// ****************************************************************

	// Eventos
	import flash.events.*;

	public class Componente2T extends Componente {

		// ------------------------------------------------
		//		Variables para manejo del ususario
		// ------------------------------------------------
		// Datos del componente para el susuario
		public var u_nombreComponente:String;
		public var u_valorComponente:Number;
		public var u_unidadesComponente:String;
		// ------------------------------------------------

		// ------------------------------------------------
		// Estructura de DATOS de "datosComponente":
		// datosComponente[0] = componente(identificador, numero_Punto_Rejilla,
		//
		//                                 xg, yg, zg)
		// datosComponente[1] = terminal1(numero_Punto_Rejilla, xg, yg, zg)
		// datosComponente[2] = terminal2(numero_Punto_Rejilla, xg, yg, zg)
		// datosComponente[n] = terminaln(numero_Punto_Rejilla, xg, yg, zg)
		// ------------------------------------------------

		// ------------------------------------------------
		// Potenciales de las "terminales" del componente
		// ------------------------------------------------
		// --- Terminal 1 ---
		public var Vnodo1:Number;	// Potencial (coord. "z")
		public var V1_t:Array;			// V1(t) (coord. "z")

		// --- Terminal 2 ---
		public var Vnodo2:Number;	// Potencial (coord. "z")
		public var V2_t:Array;			// V2(t) (coord. "z")

		public var deltaV:Number;	// Diferencia de tension

		public var corriente:Number;
		public var potencia:Number;

		// ------------------------------------------------
		// Parametros de "dibujo en 3D" del componente
		// ------------------------------------------------
		// --- Diferencia de tension normalizada ---
		private var dv:Number;
		// --- Distancia entre terminales del componente ---
		private var distancia:Number;
		// --- Angulo entre las terminales 1 y 2 ---
		private var angulo:Number;
		// --- Inclinacion del componente entre las terminales 1 y 2 ---
		private var inclinacion:Number;

		// ------------------------------------------------
```

```
// Orientacion para colocacion del componente
public var orientacion:String;
// ------------------------------------------------

// ------------------------------------------------
// Variables auxiliares para el calculo de las
// coordenadas de las terminales del componente
// ------------------------------------------------
public var numero_Punto_Rejilla_Terminal1:Number;
public var numero_Punto_Rejilla_Terminal2:Number;
// ------------------------------------------------

// ------------------------------------------------
// Modelo matematico para "Metodo Nodal Modificado"
// ------------------------------------------------
public var mna:Array;
// ------------------------------------------------

public function especificaComponente(orientacion:String):void {

	u_identificadorComponente = i_identificadorComponente;
	this.orientacion = orientacion;

	// -------------------------------
	//  Orientacion del componente
	// -------------------------------
	switch (orientacion) {
		case "N":
			// *********************************
			//                                  (1) o
			//                                      |
			//                                      |
			//                                  (2) o
			// *********************************
			rotation = 90;
			offsetHorizontal = 0;
			offsetVertical = 20;
		break;

		case "W":
			// *********************************
			//
			//                        (1) o -- o (2)
			//
			// *********************************

			rotation = 0;
			offsetHorizontal = 20;
			offsetVertical = 0;
		break;

		case "S":
			// *********************************
			//                                  (2) o
			//                                      |
			//                                      |
			//                                  (1) o
			// *********************************

			rotation = 270;
			offsetHorizontal = 0;
			offsetVertical = 20;
		break;

		case "E":
			// *********************************
			//
			//                        (2) o -- o (1)
			//
			// *********************************
```

```
                                        rotation = 180;
                                        offsetHorizontal = 20;
                                        offsetVertical = 0;
                                break;

                        }
                }

                override public function actualiza_Coordenadas_Terminales():void {

                        switch (orientacion) {
                                case "W":
                                        // ********************************
                                        //
                                        //                      (1) o -- o (2)
                                        //
                                        // ********************************
                                        numero_Punto_Rejilla_Terminal1 = (contenedorGrafico.x -
separacionEntrePuntosRejilla)/
separacionEntrePuntosRejilla +
numero_Punto_Inicial_Fila_Rejilla + 10;
                                        numero_Punto_Rejilla_Terminal2 = numero_Punto_Rejilla_Terminal1
                                                                                + longitudComponente/
separacionEntrePuntosRejilla;
                                break;

                                case "N":
                                        // ********************************
                                        //                              (1) o
                                        //                                  |
                                        //                                  |
                                        //                              (2) o
                                        // ********************************
                                        numero_Punto_Rejilla_Terminal1 =
numero_Columna_Rejilla_Componente
                                                                                +
(numero_Fila_Rejilla_Componente-1)*
(numeroPuntosAnchoRejilla + 1);
                                        numero_Punto_Rejilla_Terminal2 = numero_Punto_Rejilla_Terminal1
                                                                                +
2*(numeroPuntosAnchoRejilla+1);
                                break;

                                case "E":
                                        // ********************************
                                        //
                                        //                      (2) o -- o (1)
                                        //
                                        // ********************************
                                        numero_Punto_Rejilla_Terminal2 = (contenedorGrafico.x -
separacionEntrePuntosRejilla)
/separacionEntrePuntosRejilla +
numero_Punto_Inicial_Fila_Rejilla +10;
                                        numero_Punto_Rejilla_Terminal1 = numero_Punto_Rejilla_Terminal2
                                                                                + longitudComponente/
separacionEntrePuntosRejilla;
                                break;

                                case "S":
                                        // ********************************
                                        //                              (2) o
```

```
                                        //                                          |
                                        //                                          |
                                        //                                  (1) o
                                        // *******************************
                                        numero_Punto_Rejilla_Terminal2 =
numero_Columna_Rejilla_Componente
                                                                                    +
(numero_Fila_Rejilla_Componente-1)*
(numeroPuntosAnchoRejilla + 1);
                                        numero_Punto_Rejilla_Terminal1 = numero_Punto_Rejilla_Terminal2
                                                                                    +
2*(numeroPuntosAnchoRejilla+1);
                                break;
                        }

                        // ------------------------------------------------------------
                        // Guardado de los datos del componente en un arreglo de datos:
                        // "datosComponente"
                        // ------------------------------------------------------------
                        datosComponente[1] = new Array();
                        datosComponente[2] = new Array();

                        // --- Asignacion de coordenadas a la terminal1 del Componente ---
                        datosComponente[1].push(numero_Punto_Rejilla_Terminal1);
                        datosComponente[1].push(puntosRejilla[numero_Punto_Rejilla_Terminal1].xg);
                        datosComponente[1].push(puntosRejilla[numero_Punto_Rejilla_Terminal1].yg);
                        datosComponente[1].push(0);// coordenada "z" de la termianl1 (3)
                        // --- V1(t) (comportamiento en AC) ---
                        datosComponente[1].push(new Array());           // V1(t) (4)


                        // --- Asignacion de coordenadas a la terminal2 del Componente ---
                        datosComponente[2].push(numero_Punto_Rejilla_Terminal2);
                        datosComponente[2].push(puntosRejilla[numero_Punto_Rejilla_Terminal2].xg);
                        datosComponente[2].push(puntosRejilla[numero_Punto_Rejilla_Terminal2].yg);
                        datosComponente[2].push(0);// coordenada "z" de la termianl2 (3)
                        // --- V2(t) (comportamiento en AC) ---
                        datosComponente[2].push(new Array());           // V2(t) (4)

                        // ------------------------------------------------------------
                        //              Para fines de pruebas de funcionamiento del simulador
                        // ------------------------------------------------------------
                        //trace("Componente2T:: datosComponente[1]= "+datosComponente[1]);
                        //trace("Componente2T:: datosComponente[2]= "+datosComponente[2]);
                }

                override public function abrirPantallaEdicion(evt:MouseEvent):void {
                        if (ventanaEdicionPermitida) {
                                ventanaEdicionComponente.abrir(this);
                        }
                }

                public function inicializaMatrizNodal(mna:Array):void {
                        //  desarrollo a futuro
                }

                public function update():void {
                        nodo1 = 0;
                        nodo2 = 0;
                }

                // ------------------------------------------------------------
                //
                //                                      Seccion de SIMULACION EN AC
                //
                // ------------------------------------------------------------

                public function activarComportamiento3DComplejo():void {
```

```
                    // --- Posicion "z" del contenedor ---
                    var contenedor_z:Number;

                    removerListeners();
                    corrienteGrafica.visible = false;

                    // --- Ubicacion(t) entre nodos "z" : [0][5] ---
                    datosComponente[0][5] = new Array();
                    // --- Distancia(t) entre nodos : [0][6] ---
                    datosComponente[0][6] = new Array();
                    // --- Inclinacion(t) entre nodos : [0][7] ---
                    datosComponente[0][7] = new Array();

                    // Asignacion del "arreglo de voltajes nodales"
                    // a las terminales del componente
                    V1_t = datosComponente[1][4];
                    V2_t = datosComponente[2][4];

                    for (i in V1_t) {
                        deltaV = V1_t[i] - V2_t[i];
                        // Coloca el componente a una altura igual al punto
                        // medio de las coordenadas "z" de cada terminal
                        contenedor_z = (V1_t[i] + V2_t[i])/2;
                        // Calcula distancia geometrica entre terminales del componente
                        distancia = Math.sqrt(deltaV*deltaV + 1600);
                        // Calcula angulo de la terminal 1 a la 2
                        inclinacion = (Math.atan2(deltaV,40)/Math.PI)*180;

                        datosComponente[0][5].push(contenedor_z);
                        datosComponente[0][6].push(distancia);
                        datosComponente[0][7].push(inclinacion);
                    }
                }

                // ----------------------------------------------------------
                //
                //              Seccion "onEnterFrame" llamada desde el motor3D_ac
                //
                // ----------------------------------------------------------

                public function asignaPotencial(t:Number):void {
                    // Rota, escala y orienta el componente
                    contenedor_z = datosComponente[0][5][t];
                    distancia = datosComponente[0][6][t];
                    inclinacion = datosComponente[0][7][t];

                    this.z = contenedor_z;
                    lineaGrafica.scaleX = distancia/40;
                    contenedorGrafico.rotationY = inclinacion;
                }

                public function desactivarComportamiento3DComplejo():void {

                    this.z = 0;
                    contenedorGrafico.z = 0;
                    datosComponente[0][4] = contenedorGrafico.z;

                    contenedorGrafico.rotationY = 0;
                    lineaGrafica.scaleX = 1;

                    inicializaListeners();
                }
            }
}
```

```
package com.fabrica.productor.fabricacion
{
	// ************************************************************
	//
	//		Clase:	AlambreCorto2T (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo / 2010
	// ************************************************************
	//
	//		Clase base heredada: "Componente"
	//
	// ************************************************************

	// Eventos
	import flash.events.*;

	public class AlambreCorto2T extends Componente {

		// ------------------------------------------------
		//		Variables para manejo del ususario
		// ------------------------------------------------
		// Datos del componente para el susuario
		public var u_nombreComponente:String;
		public var u_valorComponente:Number;
		public var u_unidadesComponente:String;
		// ------------------------------------------------

		// ------------------------------------------------
		// Estructura de DATOS de "datosComponente":
		// datosComponente[0] = componente(identificador, numero_Punto_Rejilla, xg, yg, zg)
		// datosComponente[1] = terminal1(numero_Punto_Rejilla, xg, yg, zg)
		// datosComponente[2] = terminal2(numero_Punto_Rejilla, xg, yg, zg)
		// datosComponente[n] = terminaln(numero_Punto_Rejilla, xg, yg, zg)
		// ------------------------------------------------

		// ------------------------------------------------
		// Potenciales de las "terminales" del componente
		// ------------------------------------------------
		// --- Terminal 1 ---
		public var Vnodo1:Number; // Potencial (coord. "z")
		public var V1_t:Array; // V1(t) (coord. "z")

		// --- Terminal 2 ---
		public var Vnodo2:Number; // Potencial (coord. "z")
		public var V2_t:Array; // V2(t) (coord. "z")

		public var deltaV:Number; // Diferencia de tension

		public var corriente:Number;
		public var potencia:Number;

		// ------------------------------------------------
		// Parametros de "dibujo en 3D" del componente
		// ------------------------------------------------
		// Diferencia de tension normalizada
		private var dv:Number;
		// Distancia entre terminales del componente
		private var distancia:Number;
		// Angulo entre las terminales 1 y 2
		private var angulo:Number;
		// Inclinacion del componente entre las terminales 1 y 2
		private var inclinacion:Number;

		// ------------------------------------------------
		// Orientacion para colocacion del componente
```

```
public var orientacion:String;
// ------------------------------------------------

// ------------------------------------------------
// Variables auxiliares para el calculo de las
// coordenadas de las terminales del componente
// ------------------------------------------------
public var numero_Punto_Rejilla_Terminal1:Number;
public var numero_Punto_Rejilla_Terminal2:Number;
// ------------------------------------------------

// ------------------------------------------------
// Modelo matematico para "Metodo Nodal Modificado"
// ------------------------------------------------
public var mna:Array;
// ------------------------------------------------

public function especificaComponente(orientacion:String):void {

	u_identificadorComponente = i_identificadorComponente;
	this.orientacion = orientacion;

	// ------------------------------
	//  Orientacion del componente
	// ------------------------------
	switch (orientacion) {
		case "N":
			// *********************************
			//                                  (1) o
			//                                      |
			//                                  (2) o
			// *********************************
			rotation = 90;
			offsetHorizontal = 0;
			offsetVertical = 20;
		break;

		case "W":
			// *********************************
			//
			//                        (1) o - o (2)
			//
			// *********************************

			rotation = 0;
			offsetHorizontal = 20;
			offsetVertical = 0;
		break;

		case "S":
			// *********************************
			//                                  (2) o
			//                                      |
			//                                  (1) o
			// *********************************

			rotation = 270;
			offsetHorizontal = 0;
			offsetVertical = 20;
		break;

		case "E":
			// *********************************
			//
			//                        (2) o - o (1)
			//
			// *********************************
			rotation = 180;
			offsetHorizontal = 20;
			offsetVertical = 0;
```

```
                                        break;

                                }
                        }

                        override public function actualiza_Coordenadas_Terminales():void {

                                switch (orientacion) {
                                        case "W":
                                                // **********************************
                                                //
                                                //                      (1) o - o (2)
                                                //
                                                // **********************************
                                                numero_Punto_Rejilla_Terminal1 = (contenedorGrafico.x -
separacionEntrePuntosRejilla)
/separacionEntrePuntosRejilla +
numero_Punto_Inicial_Fila_Rejilla + 10;
                                                numero_Punto_Rejilla_Terminal2 = numero_Punto_Rejilla_Terminal1
                                                                                                        +
(longitudComponente/2)/separacionEntrePuntosRejilla;
                                        break;

                                        case "N":
                                                // **********************************
                                                //                              (1) o
                                                //                                  |
                                                //                              (2) o
                                                // **********************************
                                                numero_Punto_Rejilla_Terminal1 =
numero_Columna_Rejilla_Componente
                                                                                                        +
(numero_Fila_Rejilla_Componente-1)*
(numeroPuntosAnchoRejilla + 1);
                                                numero_Punto_Rejilla_Terminal2 = numero_Punto_Rejilla_Terminal1 +
(numeroPuntosAnchoRejilla+1);
                                        break;

                                        case "E":
                                                // **********************************
                                                //
                                                //                      (2) o - o (1)
                                                //
                                                // **********************************
                                                numero_Punto_Rejilla_Terminal2 = (contenedorGrafico.x -
separacionEntrePuntosRejilla)
/separacionEntrePuntosRejilla +
numero_Punto_Inicial_Fila_Rejilla + 10;
                                                numero_Punto_Rejilla_Terminal1 = numero_Punto_Rejilla_Terminal2
                                                                                                        +
(longitudComponente/2)
/separacionEntrePuntosRejilla;
                                        break;

                                        case "S":
                                                // **********************************
                                                //                              (2) o
                                                //                                  |
                                                //                              (1) o
                                                // **********************************
```

```
                                        numero_Punto_Rejilla_Terminal2 =
numero_Columna_Rejilla_Componente
                                                                                +
(numero_Fila_Rejilla_Componente-1)*
(numeroPuntosAnchoRejilla + 1);
                                        numero_Punto_Rejilla_Terminal1 = numero_Punto_Rejilla_Terminal2 +
(numeroPuntosAnchoRejilla+1);

                                        break;
                                }

                                // ---------------------------------------------------------------------------
                                // Guardado de los datos del componente en un arreglo de datos: "datosComponente"
                                // ---------------------------------------------------------------------------
                                datosComponente[1] = new Array();
                                datosComponente[2] = new Array();

                                // --- Asignacion de coordenadas a la terminal1 del Componente ----
                                datosComponente[1].push(numero_Punto_Rejilla_Terminal1);
                                datosComponente[1].push(puntosRejilla[numero_Punto_Rejilla_Terminal1].xg);
                                datosComponente[1].push(puntosRejilla[numero_Punto_Rejilla_Terminal1].yg);
                                datosComponente[1].push(0);// coordenada "z" de la termianl1

                                // --- Asignacion de coordenadas a la terminal2 del Componente ----
                                datosComponente[2].push(numero_Punto_Rejilla_Terminal2);
                                datosComponente[2].push(puntosRejilla[numero_Punto_Rejilla_Terminal2].xg);
                                datosComponente[2].push(puntosRejilla[numero_Punto_Rejilla_Terminal2].yg);
                                datosComponente[2].push(0);// coordenada "z" de la termianl2

                                // ------------------------------------------------------------
                                //              Para fines de pruebas de funcionamiento del simulador
                                // ------------------------------------------------------------
                                //trace("AlambreCorto2T:: datosComponente[1]= "+datosComponente[1]);
                                //trace("AlambreCorto2T:: datosComponente[2]= "+datosComponente[2]);
                        }

                        override public function abrirPantallaEdicion(evt:MouseEvent):void {
                                if (ventanaEdicionPermitida) {
                                        ventanaEdicionComponente.abrir(this);
                                }
                        }

                        public function inicializaMatrizNodal(mna:Array):void {
                                this.mna = mna;
                                trace(" ");
                                for (i in mna) {
                                        trace("mna["+i+"] = "+mna[i]);
                                }
                        }

                        public function update():void {
                                nodo1 = 0;
                                nodo2 = 0;
                        }

                        // ------------------------------------------------------------------
                        //                                      Seccion de SIMULACION EN AC
                        // ------------------------------------------------------------------

                        public function activarComportamiento3DComplejo():void {
                                // Posicion "z" del contenedor
                                var contenedor_z:Number;

                                removerListeners();
                                corrienteGrafica.visible = false;

                                datosComponente[0][5] = new Array(); // Ubicacion(t) entre nodos "z" // [0][5]
```

```
                    datosComponente[0][6] = new Array(); // Distancia(t) entre nodos        // [0][6]
                    datosComponente[0][7] = new Array(); // Inclinacion(t) entre nodos      // [0][7]

                    // Asignacion del "arreglo de voltajes nodales"
                    // a las terminales del componente
                    V1_t = datosComponente[1][4];
                    V2_t = datosComponente[2][4];

                    for (i in V1_t) {
                        deltaV = V1_t[i] - V2_t[i];
                        // Coloca el componente a una altura igual al punto
                        // medio de las coordenadas "z" de cada terminal
                        contenedor_z = (V1_t[i] + V2_t[i])/2;
                        // Calcula distancia geometrica entre terminales del componente
                        distancia = Math.sqrt(deltaV*deltaV + 1600);
                        // Calcula angulo de la terminal 1 a la 2
                        inclinacion = (Math.atan2(deltaV,40)/Math.PI)*180;

                        datosComponente[0][5].push(contenedor_z);
                        datosComponente[0][6].push(distancia);
                        datosComponente[0][7].push(inclinacion);
                    }
                }

                // ----------------------------------------------------------
                //
                //              Seccion "onEnterFrame" llamada desde el motor3D_ac
                //
                // ----------------------------------------------------------

                public function asignaPotencial(t:Number):void {
                    // Rota, escala y orienta el componente
                    contenedor_z = datosComponente[0][5][t];
                    distancia = datosComponente[0][6][t];
                    inclinacion = datosComponente[0][7][t];

                    this.z = contenedor_z;
                    lineaGrafica.scaleX = distancia/40;
                    contenedorGrafico.rotationY = inclinacion;
                }

                public function desactivarComportamiento3DComplejo():void {

                    this.z = 0;
                    contenedorGrafico.z = 0;
                    datosComponente[0][4] = contenedorGrafico.z;

                    contenedorGrafico.rotationY = 0;
                    lineaGrafica.scaleX = 1;

                    inicializaListeners();
                }

            }
}
```

package com.fabrica.productor.fabricacion

```
{
	// ****************************************************************
	//
	//		Clase:	Componente3T (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                                  mayo / 2010
	// ****************************************************************
	//
	//		Clase base heredada: "Componente"
	//
	// ****************************************************************

	// Eventos
	import flash.events.*;

	public class Componente3T extends Componente {

		// ------------------------------------------------
		//		Variables para manejo del ususario
		// ------------------------------------------------
		// Datos del componente para el susuario
		public var u_nombreComponente:String;
		public var u_valorComponente:Number;
		public var u_unidadesComponente:String;
		// ------------------------------------------------

		// ------------------------------------------------
		// Estructura de DATOS de "datosComponente":
		// datosComponente[0] = componente(identificador, numero_Punto_Rejilla,
		//
		//                                 xg, yg, zg)
		// datosComponente[1] = terminal1(numero_Punto_Rejilla, xg, yg, zg)
		// datosComponente[2] = terminal2(numero_Punto_Rejilla, xg, yg, zg)
		// datosComponente[n] = terminaln(numero_Punto_Rejilla, xg, yg, zg)
		// ------------------------------------------------

		// ------------------------------------------------
		// Potenciales de las "terminales" del componente
		// ------------------------------------------------
		// --- Terminal 1 ---
		public var Vnodo1:Number; // Potencial (coord. "z")
		public var V1_t:Array; // V1(t) (coord. "z")

		// --- Terminal 2 ---
		public var Vnodo2:Number; // Potencial (coord. "z")
		public var V2_t:Array; // V2(t) (coord. "z")

		// --- Terminal 3 ---
		public var Vnodo3:Number; // Potencial (coord. "z")
		public var V3_t:Array; // V3(t) (coord. "z")

		public var deltaV:Number; // Diferencia de tension
		public var corriente:Number;
		public var potencia:Number;
		// ------------------------------------------------
		// Orientacion para colocacion del componente
		public var orientacion:String;
		// ------------------------------------------------

		// ------------------------------------------------
		// Variables auxiliares para el calculo de las
		// coordenadas de las terminales del componente
		// ------------------------------------------------
		public var numero_Punto_Rejilla_Terminal1:Number;
		public var numero_Punto_Rejilla_Terminal2:Number;
		public var numero_Punto_Rejilla_Terminal3:Number;
```

```
		// ----------------------------------------------

		// ----------------------------------------------
		// Modelo matematico para "Metodo Nodal Modificado"
		// ----------------------------------------------
		public var mna:Array;
		// ----------------------------------------------

		public function especificaComponente(orientacion:String):void {

			u_identificadorComponente = i_identificadorComponente;
			this.orientacion = orientacion;

			// -------------------------------
			//  Orientacion del componente
			// -------------------------------
			offsetHorizontal = longitudComponente/2;
			offsetVertical = longitudComponente/2;

			switch (orientacion) {
				case "W":
					// *********************************
					//                                          o (2)
					//                    (1) o
					//                                          o (3)
					// *********************************
					rotation = 0;
				break;

				case "S":
					// *********************************
					//          (2)    o        o (3)
					//                             o (1)
					//
					// *********************************
					rotation = 270;
				break;

				default:
					trace(" ");
					trace("*********************************************************");
					trace("*    ERROR: COMPONENTE NO SOPORTADO
	*");
					trace("*********************************************************");
				break;
			}
		}

		override public function actualiza_Coordenadas_Terminales():void {

			switch (orientacion) {
				case "E":
					// *********************************
					//                    (-)(2) o
					//                                                    o (3)(Vo)
					//                    (+)(1) o
					// *********************************
					numero_Punto_Rejilla_Terminal1 = datosComponente[0][1] +
numeroPuntosAnchoRejilla;
					numero_Punto_Rejilla_Terminal2 = datosComponente[0][1] -
numeroPuntosAnchoRejilla - 2;
					numero_Punto_Rejilla_Terminal3 = datosComponente[0][1] + 1;
				break;

				case "W":
					// *********************************
					//                                          o (2)
					//                    (1) o
```

```
                                        //                                          o (3)
                                        // *********************************
                                        // NPN - Base / CH-N - Gate / OPAMP - Output
                                        // PNP - Base / CH-P - Gate / OPAMP - Output
                                        numero_Punto_Rejilla_Terminal1 = (contenedorGrafico.x -
separacionEntrePuntosRejilla)
/separacionEntrePuntosRejilla +
numero_Punto_Inicial_Fila_Rejilla + 10;
                                        // NPN - Colector / CH-N - Drain / OPAMP - Inversora
                                        // PNP - Colector / CH-P - Drain / OPAMP - No-Inversora
                                        numero_Punto_Rejilla_Terminal2 = numero_Punto_Rejilla_Terminal1 -
numeroPuntosAnchoRejilla;
                                        // NPN - Emisor / CH-N - Source / OPAMP - No-Inversora
                                        // NPN - Emisor / CH-N - Source / OPAMP - Inversora
                                        numero_Punto_Rejilla_Terminal3 = numero_Punto_Rejilla_Terminal1 +
numeroPuntosAnchoRejilla + 2;
                                    break;

                                    case "S":
                                        // *********************************
                                        //              (2)       o             o (3)
                                        //                                       o (1)
                                        //
                                        // *********************************
                                        // NPN - Base / CH-P - Gate / OPAMP - Output
                                        numero_Punto_Rejilla_Terminal1 = (contenedorGrafico.x -
separacionEntrePuntosRejilla)
/separacionEntrePuntosRejilla +
numero_Punto_Inicial_Fila_Rejilla
                                                                                                    +
numeroPuntosAnchoRejilla + 2 + 10;
                                        // NPN - Colector / CH-P - Drain / OPAMP - Inversora
                                        numero_Punto_Rejilla_Terminal2 = numero_Punto_Rejilla_Terminal1 -
numeroPuntosAnchoRejilla - 2;
                                        // NPN - Emisor      / CH-P - Source      / OPAMP - No-Inversora
                                        numero_Punto_Rejilla_Terminal3 = numero_Punto_Rejilla_Terminal1 -
numeroPuntosAnchoRejilla;
                                    break;

                                    default:
                                        trace(" ");
                                        trace("*******************************************************");
                                        trace("*    ERROR: COMPONENTE NO SOPORTADO
                *");
                                        trace("*******************************************************");
                                    break;
                            }

                            // ------------------------------------------------------------
                            // Guardado de los datos del componente en un arreglo de datos:
                            // "datosComponente"
                            // ------------------------------------------------------------
                            datosComponente[1] = new Array();
                            datosComponente[2] = new Array();
                            datosComponente[3] = new Array();

                            // --- Asignacion de coordenadas a la terminal1 del Componente ---
                            datosComponente[1].push(numero_Punto_Rejilla_Terminal1);
                            datosComponente[1].push(puntosRejilla[numero_Punto_Rejilla_Terminal1].x);
```

```
                        datosComponente[1].push(puntosRejilla[numero_Punto_Rejilla_Terminal1].y);
                        datosComponente[1].push(0);// coordenada "z" de la termianl1 (3)
                        // --- V1(t) (comportamiento en AC) ---
                        datosComponente[1].push(new Array());           // [+] V1(t) (4)

                        // --- Asignacion de coordenadas a la terminal2 del Componente ---
                        datosComponente[2].push(numero_Punto_Rejilla_Terminal2);
                        datosComponente[2].push(puntosRejilla[numero_Punto_Rejilla_Terminal2].x);
                        datosComponente[2].push(puntosRejilla[numero_Punto_Rejilla_Terminal2].y);
                        datosComponente[2].push(0);// coordenada "z" de la termianl2 (3)
                        // --- V2(t) (comportamiento en AC) ---
                        datosComponente[2].push(new Array());           // [-] V2(t) (4)

                        // --- Asignacion de coordenadas a la terminal3 del Componente ---
                        datosComponente[3].push(numero_Punto_Rejilla_Terminal3);
                        datosComponente[3].push(puntosRejilla[numero_Punto_Rejilla_Terminal3].x);
                        datosComponente[3].push(puntosRejilla[numero_Punto_Rejilla_Terminal3].y);
                        datosComponente[3].push(0);// coordenada "z" de la termianl3 (3)
                        // --- V2(t) (comportamiento en AC) ---
                        datosComponente[3].push(new Array());           // [Vo] V3(t) (4)

                        // ------------------------------------------------------------
                        //              Para fines de pruebas de funcionamiento del simulador
                        // ------------------------------------------------------------
                        //trace("Componente3TF:: [x] datosComponente[0]= "+datosComponente[0]);
                        //trace("Componente3TF:: [+] datosComponente[1]= "+datosComponente[1]);
                        //trace("Componente3TF:: [-] datosComponente[2]= "+datosComponente[2]);
                        //trace("Componente3TF:: [Vo] datosComponente[3]= "+datosComponente[3]);

                }

                override public function abrirPantallaEdicion(evt:MouseEvent):void {
                        if (ventanaEdicionPermitida) {
                                ventanaEdicionComponente.abrir(this);
                        }
                }

                public function update():void {

                }
        }
}
```

```
package com.fabrica.productor.fabricacion
{
	// ************************************************************
	//
	//	Clase ABSTRACTA:	FabricarComponente (FABRICA)
	//
	//	Patron de Programacion:
	//
	//		"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	// Esta clase constituye la definicion de la clase base para el
	// proceso de FABRICACION de todos los componentes utilizados en
	// el simulador
	//
	//	Clases derivadas:
	//
	//			- FabricarTierra		--> tierra
	//			- FabricarAlambre1		--> alambre largo (40 pixeles)
	//			- FabricarAlambre2		--> alambre corto (20 pixeles)
	//			- FabricarResistencia	--> resistencia
	//			- FabricarCapacitor		--> capacitor
	//			- FabricarVDC			--> fuente de voltaje DC
	//			- FabricarVAC			--> fuente de voltaje AC
	//			- FabricarIDC			--> fuente de corriente DC
	//			- FabricarOpAmp				--> amplificador
operacional
	//
	// ************************************************************

	// Flash principal
	import flash.display.Sprite;

	// Error al intentar generar un objeto de esta clase abstracta
	import flash.errors.IllegalOperationError;

	// Clase ABSTRACTA (debe ser "extendida"
	// y no intentar "generar objetos" de ella)
	public class FabricarComponente {

		public function Fabricar(target:Sprite, parametros:Object,

	orientacion:String):void {

			var componente:Componente = this.CrearComponente();
			componente.inicializaParametros(parametros,target);
			componente.especificaComponente(orientacion);
			componente.colocacionInicial();
			target.addChild(componente);
		}

		protected function CrearComponente():Componente {
			// ------------------------------------------------------------
```

```
                        //              Metodo ABSTRACTO: se debe "definir" en una subclase
                        // ------------------------------------------------------------
                        throw new IllegalOperationError("Metodo abtracto: debe ser definido en una
subclase");
                        return null;
                }

        }
}
```

```
package com.fabrica.productor.componentes.alambre
{
	// ************************************************************
	//
	//		Clase:  FabricarAlambre1 (FABRICA)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarAlambre1 extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new Alambre1;
		}

	}
}
```

```
package com.fabrica.productor.componentes.alambre
{
	// ************************************************************
	//
	//		Clase:  FabricarAambre2 (FABRICA)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarAlambre2 extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new Alambre2;
		}

	}
}
```

```
package com.fabrica.productor.componentes.capacitor
{
	// ************************************************************
	//
	//		Clase:  FabricarCapacitor (FABRICA)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarCapacitor extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new Capacitor;
		}

	}
}
```

```
package com.fabrica.productor.componentes.resistencia
{
	// ************************************************************
	//
	//	Clase:  FabricarResistencia (FABRICA)
	//
	//	Patron de Programacion:
	//
	//		"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//	Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarResistencia extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new Resistencia;
		}

	}
}
```

```
package com.fabrica.productor.componentes.fuentesVoltaje
{
	// ************************************************************
	//
	//		Clase:	FabricarVDC (FABRICA)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarVDC extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new FuenteVDC;
		}

	}
}
```

```
package com.fabrica.productor.componentes.fuentesVoltaje
{
	// ************************************************************
	//
	//		Clase:  FabricarVAC (FABRICA)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarVAC extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new FuenteVAC;
		}

	}
}
```

```
package com.fabrica.productor.componentes.fuentesCorriente
{
	// ************************************************************
	//
	//		Clase:  FabricarIDC (FABRICA)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarIDC extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new FuenteIDC;
		}

	}
}
```

```
package com.fabrica.productor.componentes.opamp
{
	// ************************************************************
	//
	//		Clase:	FabricarOpAmp (FABRICA)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarOpAmp extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new OpAmp;
		}

	}
}
```

```
package com.fabrica.productor.componentes.conexion
{
	// ************************************************************
	//
	//		Clase:	FabricarTierra (FABRICA)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "FabricarComponente"
	//
	// ************************************************************

	// Establece la FUNCIONALIDAD del componente
	import com.fabrica.productor.fabricacion.Componente;
	// Proceso de FABRICACION del componente
	import com.fabrica.productor.fabricacion.FabricarComponente;

	public class FabricarTierra extends FabricarComponente {

		override protected function CrearComponente():Componente {
			return new Tierra;
		}

	}
}
```

```
package com.fabrica.productor.componentes.alambre
{
	// ************************************************************
	//
	//	Clase:  Alambre1 (PRODUCTO)		[Alambre largo: 40 pixeles]
	//
	//	Patron de Programacion:
	//
	//		"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//	Clase base heredada: "Componente2T"
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Clase base
	import com.fabrica.productor.fabricacion.Componente2T;

	public class Alambre1 extends Componente2T {

		// ------------------------------------------------------------
		//
		//		Seccion de ESPECIFICACION del alambre1 (largo)
		//
		// ------------------------------------------------------------

		public function Alambre1():void {

			// ------------------------------------------------------------
			//		Constructor del objeto "Alambre1"
			// ------------------------------------------------------------

			// --- Genera dibujo del componente ---
			// Capa 0 : "contenedor"
			contenedorGrafico = new MovieClip();
			addChild(contenedorGrafico);

			// Capa 1 : "fondo"
			fondoGrafico = new mc_FondoLargo();
			contenedorGrafico.addChild(fondoGrafico);

			// Capa 2 : "simbolo"
			simboloGrafico = new mc_ALargo();
			lineaGrafica = simboloGrafico;
			contenedorGrafico.addChild(simboloGrafico);

			// Capa 3 : "corriente"
			corrienteGrafica = new mc_Corriente();
			contenedorGrafico.addChild(corrienteGrafica);
```

```
            // --- Inicializa parametros ---
            i_tipoComponente = "W";
            u_nombreComponente = "Alambre";
            u_valorComponente = 0;                    // 0 ohms
            u_unidadesComponente = "ohms";

            // --- No permite abrir ventana de edicion ---
            ventanaEdicionPermitida = false;
        }

        public function ventanaInformativa():Array {

            // ------------------------------------------------------
            //                          Ventana informativa del objeto "Alambre1"
            // ------------------------------------------------------
            var infoVentana:Array = new Array();
            infoVentana[0] = u_nombreComponente;
            infoVentana[1] = new Array();
            infoVentana[1] = ["Identificador:", , ];
            infoVentana[2] = new Array();
            infoVentana[2] = ["Valor:",u_valorComponente,u_unidadesComponente];

            return infoVentana;
        }

        // ------------------------------------------------------
        //
        //              Seccion de EVALUACION de VOLTAJES y CORRIENTES
        //
        // ------------------------------------------------------

        public function asignarVoltajesNodales():void {

            // --- Asignacion de voltajes nodales ---
            Vnodo1 = datosComponente[1][3];
            Vnodo2 = datosComponente[2][3];
            deltaV = Vnodo2 - Vnodo1;
        }

        public function asignarCorrientesRama(corriente:Number):void {

            // --- Asignacion de corrientes de rama ---
            this.corriente = corriente;
        }

        // ------------------------------------------------------
        //
        //              Seccion de ACTIVACION DE COMPORTAMIENTO EN 3D
        //
        // ------------------------------------------------------

        public function activarComportamiento3D(factorEscala3D:Number,

escala3D_corriente:Object):void {
            // --- Inicializa componente ---
            colocarComponenteEspacio3D(factorEscala3D);
```

```
                // --- Activar corriente ---
                activarCorriente(escala3D_corriente);
        }

        public function colocarComponenteEspacio3D(factorEscala3D:Number):void {

                removerListeners();
                // Coloca el componente a una altura igual al punto medio de las
                // coordenadas "z" de cada terminal y aplica el factor de escala
                // para normalizar la altura del componente
                contenedorGrafico.z = factorEscala3D*Vnodo1;
                // Actualiza la coordenada "z" del componente
                datosComponente[0][4] = contenedorGrafico.z;

                // Aplica el factor de escala en 3D para normalizar la diferencia
                // de voltajes entre las terminales del componente
                dv = factorEscala3D*deltaV;
                // Calcula distancia geometrica entre terminales del componente
                distancia = Math.sqrt(dv*dv + 1600);
                // Calcula angulo de la terminal 1 a la 2
                angulo = (Math.atan2(dv,40)/Math.PI)*180;

                // Rota, escala y orienta el componente de acuerdo a los resultados
                contenedorGrafico.rotationY = angulo;
                lineaGrafica.scaleX = distancia/40;
        }

        public function activarCorriente(escala3D_corriente:Object):void {

                // --- CORRIENTE ---
                var valorCorriente:Number;
                if (escala3D_corriente.lineal) {
                        // Representacion en ESCALA LINEAL
                        valorCorriente = 10;
                } else {
                        // Representacion en ESCALA LOGARITMICA
                        var corrienteMinima:Number =
escala3D_corriente.corrienteMinima;
                        var factor_conversion:Number =
escala3D_corriente.factor_conversion;
                        valorCorriente =
Math.ceil(factor_conversion*LN_LOG*Math.log(Math.abs(corriente)/corrienteMinima))+1;
                }

                if (Math.abs(corriente) > 1e-10) {
                        corrienteGrafica.rotationY = 0;
                        if (corriente<0) {
                                corrienteGrafica.rotationY = 180;
                        }
                        corrienteGrafica.scaleX = distancia/40;
                        corrienteGrafica.alpha = 0.9;
                        lineaGrafica.alpha = valorCorriente/10;
                        fondoGrafico.alpha = valorCorriente/10;
                        corrienteGrafica.gotoAndPlay("i"+valorCorriente.toString());
                }
```

```
                corrienteGrafica.visible = true;
            }

            public function desactivarComportamiento3D():void {

                contenedorGrafico.z = 0;
                datosComponente[0][4] = contenedorGrafico.z;

                contenedorGrafico.rotationY = 0;
                lineaGrafica.scaleX = 1;
                lineaGrafica.alpha = 1;
                fondoGrafico.alpha = 1;
                corrienteGrafica.scaleX = 1;
                corrienteGrafica.gotoAndStop("i0");

                inicializaListeners();
            }

        }
}
```

```
package com.fabrica.productor.componentes.alambre
{
	// ************************************************************
	//
	//	Clase:	Alambre2 (PRODUCTO)		[Alambre corto: 20 pixeles]
	//
	//	Patron de Programacion:
	//
	//		"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//	Clase base heredada: "Componente2T"
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Clase base
	import com.fabrica.productor.fabricacion.AlambreCorto2T;

	public class Alambre2 extends AlambreCorto2T {

		// ------------------------------------------------------------
		//
		//		Seccion de ESPECIFICACION del alambre2 (corto)
		//
		// ------------------------------------------------------------

		public function Alambre2():void {

			// ------------------------------------------------------------
			//			Constructor del objeto "Alambre2"
			// ------------------------------------------------------------

			// --- Genera dibujo del componente ---
			// Capa 0 : "contenedor"
			contenedorGrafico = new MovieClip();
			addChild(contenedorGrafico);

			// Capa 1 : "fondo"
			fondoGrafico = new mc_FondoCorto();
			contenedorGrafico.addChild(fondoGrafico);

			// Capa 2 : "simbolo"
			simboloGrafico = new mc_ACorto();
			lineaGrafica = simboloGrafico;
			contenedorGrafico.addChild(simboloGrafico);

			// Capa 2 : "corriente"
			corrienteGrafica = new mc_Corriente2();
			contenedorGrafico.addChild(corrienteGrafica);
```

```
                // --- Inicializa parametros ---
                i_tipoComponente = "W2";
                u_nombreComponente = "Alambre";
                u_valorComponente = 0;                  // 0 ohms
                u_unidadesComponente = "ohms";

                // --- No permite abrir ventana de edicion ---
                ventanaEdicionPermitida = false;
        }

        public function ventanaInformativa():Array {

                // ------------------------------------------------------
                //                      Ventana informativa del objeto "Alambre2"
                // ------------------------------------------------------
                var infoVentana:Array = new Array();
                infoVentana[0] = u_nombreComponente;
                infoVentana[1] = new Array();
                infoVentana[1] = ["Identificador:", , ];
                infoVentana[2] = new Array();
                infoVentana[2] = ["Valor:",u_valorComponente,u_unidadesComponente];

                return infoVentana;
        }

        // ------------------------------------------------------
        //
        //          Seccion de EVALUACION de VOLTAJES y CORRIENTES
        //
        // ------------------------------------------------------

        public function asignarVoltajesNodales():void {

                // --- Asignacion de voltajes nodales ---
                Vnodo1 = datosComponente[1][3];
                Vnodo2 = datosComponente[2][3];
                deltaV = Vnodo2 - Vnodo1;
        }

        public function asignarCorrientesRama(corriente:Number):void {

                // --- Asignacion de corrientes de rama ---
                this.corriente = corriente;
        }

        // ------------------------------------------------------
        //
        //          Seccion de ACTIVACION DE COMPORTAMIENTO EN 3D
        //
        // ------------------------------------------------------

        public function
activarComportamiento3D(factorEscala3D:Number,escala3D_corriente:Object):void {
                // Inicializa componente
                colocarComponenteEspacio3D(factorEscala3D);
                // Activar corriente
```

```
                activarCorriente(escala3D_corriente);
        }

        public function colocarComponenteEspacio3D(factorEscala3D:Number):void {

                removerListeners();
                // Coloca el componente a una altura igual al punto medio de las
                // coordenadas "z" de cada terminal y aplica el factor de escala
                // para normalizar la altura del componente
                contenedorGrafico.z = factorEscala3D*Vnodo1;
                // Actualiza la coordenada "z" del componente
                datosComponente[0][4] = contenedorGrafico.z;

                // Aplica el factor de escala en 3D para normalizar la diferencia
                // de voltajes entre las terminales del componente
                dv = factorEscala3D*deltaV;
                // Calcula distancia geometrica entre terminales del componente
                distancia = Math.sqrt(dv*dv + 1600);
                // Calcula angulo de la terminal 1 a la 2
                angulo = (Math.atan2(dv,40)/Math.PI)*180;

                // Rota, escala y orienta el componente de acuerdo a los resultados
                contenedorGrafico.rotationY = angulo;
                lineaGrafica.scaleX = distancia/40;
        }

        public function activarCorriente(escala3D_corriente:Object):void {

                // --- CORRIENTE ---
                var valorCorriente:Number;
                if (escala3D_corriente.lineal) {
                        // Representacion en ESCALA LINEAL
                        valorCorriente = 10;
                } else {
                        // Representacion en ESCALA LOGARITMICA
                        var corrienteMinima:Number =
escala3D_corriente.corrienteMinima;
                        var factor_conversion:Number =
escala3D_corriente.factor_conversion;
                        valorCorriente =
Math.ceil(factor_conversion*LN_LOG*Math.log(Math.abs(corriente)/corrienteMinima))+1;
                }

                if (Math.abs(corriente) > 1e-10) {
                        corrienteGrafica.rotationY = 0;
                        corrienteGrafica.x = -10;           // Para centrar la corriente
                        if (corriente<0) {
                                corrienteGrafica.rotationY = 180;
                        }
                        corrienteGrafica.scaleX = distancia/40;
                        corrienteGrafica.alpha = 0.9;
                        lineaGrafica.alpha = valorCorriente/10;
                        fondoGrafico.alpha = valorCorriente/10;
                        corrienteGrafica.gotoAndPlay("i"+valorCorriente.toString());
                }
```

```
                corrienteGrafica.visible = true;
            }

            public function desactivarComportamiento3D():void {

                contenedorGrafico.z = 0;
                datosComponente[0][4] = contenedorGrafico.z;

                contenedorGrafico.rotationY = 0;
                lineaGrafica.scaleX = 1;
                lineaGrafica.alpha = 1;
                fondoGrafico.alpha = 1;
                corrienteGrafica.scaleX = 1;
                corrienteGrafica.gotoAndStop("i0");

                inicializaListeners();
            }

        }
}
```

```
package com.fabrica.productor.componentes.capacitor
{
	// ************************************************************
	//
	//		Clase:  Capacitor (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "Componente2T"
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Clase base
	import com.fabrica.productor.fabricacion.Componente2T;

	public class Capacitor extends Componente2T {

		public var C:Number;

		// ------------------------------------------------------------
		//
		//                          Seccion de ESPECIFICACION del capacitor
		//
		// ------------------------------------------------------------

		public function Capacitor():void {

			// ------------------------------------------------------------
			//                          Constructor del objeto "Capacitor"
			// ------------------------------------------------------------

			// --- Genera dibujo del componente ---
			// Capa 0 : "contenedor"
			contenedorGrafico = new MovieClip();
			addChild(contenedorGrafico);

			// Capa 1 : "linea"
			lineaGrafica = new mc_Linea2();
			contenedorGrafico.addChild(lineaGrafica);

			// Capa 2 : "corriente"
			corrienteGrafica = new mc_Corriente();
			corrienteGrafica.z = 0;
			corrienteGrafica.visible = false;
			contenedorGrafico.addChild(corrienteGrafica);

			// Capa 3 : "simbolo"
```

```
            simboloGrafico = new mc_C();
            contenedorGrafico.addChild(simboloGrafico);

            // --- Inicializa parametros ---
            i_tipoComponente = "C";
            u_nombreComponente = "Capacitor";
            u_valorComponente = 100E-6;             // 100 uF
            u_unidadesComponente = "Farad";
            G = u_valorComponente;                  // 2*PI*f = 1

            // --- Permite abrir ventana de edicion ---
            ventanaEdicionPermitida = true;
        }

        public function ventanaInformativa():Array {

            // ------------------------------------------------------------
            //                          Ventana informativa del objeto "Capacitor"
            // ------------------------------------------------------------
            var infoVentana:Array = new Array();
            infoVentana[0] = u_nombreComponente;
            infoVentana[1] = new Array();
            infoVentana[1] = ["Identificador:", , ];
            infoVentana[2] = new Array();
            infoVentana[2] = ["Valor:",u_valorComponente,u_unidadesComponente];

            return infoVentana;
        }

        // ------------------------------------------------------------
        //
        //              Seccion de EVALUACION de VOLTAJES y CORRIENTES
        //
        // ------------------------------------------------------------

        public function asignarVoltajesNodales():void {
            // Asignacion de voltajes nodales
            Vnodo1 = datosComponente[1][3];
            Vnodo2 = datosComponente[2][3];
            deltaV = Vnodo2 - Vnodo1;

            // Actualiza valor del componente
            C = u_valorComponente;
        }

        public function asignarCorrientesRama(corriente:Number):void {
            this.corriente = corriente;

            // Calculo de la potencia reactiva
            potencia = deltaV * corriente;
        }

        // ------------------------------------------------------------
        //
        //              Seccion de ACTIVACION DE COMPORTAMIENTO EN 3D
        //
```

```
		// ----------------------------------------------------------

		public function activarComportamiento3D(factorEscala3D:Number,

	escala3D_corriente:Object):void {
			// Inicializa componente
			colocarComponenteEspacio3D(factorEscala3D);
			// Activar corriente
			activarCorriente(escala3D_corriente);
		}

		public function colocarComponenteEspacio3D(factorEscala3D:Number):void {

			removerListeners();
			// Coloca el componente a una altura igual al punto medio de las
			// coordenadas "z" de cada terminal y aplica el factor de escala
			// para normalizar la altura del componente
			contenedorGrafico.z = factorEscala3D*(Vnodo1 + Vnodo2)/2;
			// Actualiza la coordenada "z" del componente
			datosComponente[0][4] = contenedorGrafico.z;

			// Aplica el factor de escala en 3D para normalizar la diferencia
			// de voltajes entre las terminales del componente
			dv = factorEscala3D*deltaV;
			// Calcula distancia geometrica entre terminales del componente
			distancia = Math.sqrt(dv*dv + 1600);
			// Calcula angulo de la terminal 1 a la 2
			angulo = (Math.atan2(dv,40)/Math.PI)*180;

			// Rota, escala y orienta el componente de acuerdo a los resultados
			contenedorGrafico.rotationY = -angulo;
			lineaGrafica.scaleX = distancia/40;
		}

		public function activarCorriente(escala3D_corriente:Object):void {

			// --- CORRIENTE ---
			var valorCorriente:Number;
			if (escala3D_corriente.lineal) {
				// Representacion en ESCALA LINEAL
				valorCorriente = 10;
			} else {
				// Representacion en ESCALA LOGARITMICA
				var corrienteMinima:Number =
escala3D_corriente.corrienteMinima;
				var factor_conversion:Number =
escala3D_corriente.factor_conversion;
				valorCorriente = Math.ceil(factor_conversion*LN_LOG*

Math.log(Math.abs(corriente)/corrienteMinima))+1;
			}

			if (Math.abs(deltaV) > 1e-10) {
				corrienteGrafica.scaleX = distancia/40;
				corrienteGrafica.alpha = 0.9;
				lineaGrafica.alpha = valorCorriente/10;
```

```
                        corrienteGrafica.gotoAndPlay("i"+valorCorriente.toString());
                        corrienteGrafica.rotationY = 0;
                        if (deltaV<0) {
                                corrienteGrafica.rotationY = 180;
                        }
                }
        }

        public function desactivarComportamiento3D():void {

                contenedorGrafico.z = 0;
                datosComponente[0][4] = contenedorGrafico.z;

                contenedorGrafico.rotationY = 0;
                lineaGrafica.scaleX = 1;
                lineaGrafica.alpha = 1;
                corrienteGrafica.scaleX = 1;
                corrienteGrafica.gotoAndStop("i0");

                inicializaListeners();
        }
    }
}
```

```
package com.fabrica.productor.componentes.resistencia
{
	// ************************************************************
	//
	//		Clase:  Resistencia (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "Componente2T"
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Clase base
	import com.fabrica.productor.fabricacion.Componente2T;

	public class Resistencia extends Componente2T {

		public var R:Number;

		// ------------------------------------------------------------
		//
		//				Seccion de ESPECIFICACION de la resistencia
		//
		// ------------------------------------------------------------

		public function Resistencia():void {

			// ------------------------------------------------------------
			//				Constructor del objeto "Resistencia"
			// ------------------------------------------------------------

			// --- Genera dibujo del componente ---
			// Capa 0 : "contenedor"
			contenedorGrafico = new MovieClip();
			addChild(contenedorGrafico);

			// Capa 1 : "linea"
			lineaGrafica = new mc_Linea();
			contenedorGrafico.addChild(lineaGrafica);

			// Capa 2 : "corriente"
			corrienteGrafica = new mc_Corriente();
			corrienteGrafica.z = 0;
			contenedorGrafico.addChild(corrienteGrafica);

			// Capa 3 : "simbolo"
			simboloGrafico = new mc_R();
```

```
            contenedorGrafico.addChild(simboloGrafico);

            // --- Inicializa parametros ---
            i_tipoComponente = "R";
            u_nombreComponente = "Resistencia";
            u_valorComponente = 1000;               // 1000 ohms
            u_unidadesComponente = "ohms";

            // --- Permite abrir ventana de edicion ---
            ventanaEdicionPermitida = true;
}

public function ventanaInformativa():Array {

            // ------------------------------------------------------------
            //                          Ventana informativa del objeto "Resistencia"
            // ------------------------------------------------------------
            var infoVentana:Array = new Array();
            infoVentana[0] = u_nombreComponente;
            infoVentana[1] = new Array();
            infoVentana[1] = ["Identificador:", , ];
            infoVentana[2] = new Array();
            infoVentana[2] = ["Valor:",u_valorComponente,u_unidadesComponente];

            return infoVentana;
}

// ------------------------------------------------------------
//
//              Seccion de EVALUACION de VOLTAJES y CORRIENTES
//
// ------------------------------------------------------------

public function asignarVoltajesNodales():void {

            // --- Asignacion de voltajes nodales ---
            Vnodo1 = datosComponente[1][3];
            Vnodo2 = datosComponente[2][3];
            deltaV = Vnodo1 - Vnodo2;

            // --- Actualiza valor del componente ---
            R = u_valorComponente;

            // --- Calculo de la corriente ---
            if (Math.abs(deltaV) > 1e-10) {
                        corriente = deltaV/R;           // Ley de Ohm
            } else {
                        corriente = 0;
            }

            // --- Calculo de la potencia ---
            potencia = deltaV * corriente;
}

// ------------------------------------------------------------
//
```

```
		//			Seccion de ACTIVACION DE COMPORTAMIENTO EN 3D
		//
		// ------------------------------------------------------

		public function activarComportamiento3D(factorEscala3D:Number,

	escala3D_corriente:Object):void {
			// --- Inicializa componente ---
			colocarComponenteEspacio3D(factorEscala3D);
			// --- Activar corriente ---
			activarCorriente(escala3D_corriente);
		}

		public function colocarComponenteEspacio3D(factorEscala3D:Number):void {

			removerListeners();
			// Coloca el componente a una altura igual al punto medio de las
			// coordenadas "z" de cada terminal y aplica el factor de escala
			// para normalizar la altura del componente
			contenedorGrafico.z = factorEscala3D*(Vnodo1 + Vnodo2)/2;
			// Actualiza la coordenada "z" del componente
			datosComponente[0][4] = contenedorGrafico.z;

			calculaDistanciaAngulo(factorEscala3D);

			// Rota, escala y orienta el componente de acuerdo a los resultados
			contenedorGrafico.rotationY = -angulo;
			lineaGrafica.scaleX = distancia/40;
		}

		public function calculaDistanciaAngulo(factorEscala3D:Number):void {

			deltaV = Vnodo2 - Vnodo1;
			// Aplica el factor de escala en 3D para normalizar la diferencia
			// de voltajes entre las terminales del componente
			dv = factorEscala3D*deltaV;
			// Calcula distancia geometrica entre terminales del componente
			distancia = Math.sqrt(dv*dv + 1600);
			// Calcula angulo de la terminal 1 a la 2
			angulo = (Math.atan2(dv,40)/Math.PI)*180;
		}

		private function activarCorriente(escala3D_corriente:Object):void {

			// --- CORRIENTE ---
			var valorCorriente:Number;
			if (escala3D_corriente.lineal) {
				// Representacion en ESCALA LINEAL
				valorCorriente = 10;
			} else {
				// Representacion en ESCALA LOGARITMICA
				var corrienteMinima:Number =
escala3D_corriente.corrienteMinima;
				var factor_conversion:Number =
escala3D_corriente.factor_conversion;
				valorCorriente = Math.ceil(factor_conversion*LN_LOG*
```

```
Math.log(Math.abs(corriente)/corrienteMinima))+1;
                        }

                        if (Math.abs(deltaV) > 1e-10) {
                                corrienteGrafica.scaleX = distancia/40;
                                corrienteGrafica.alpha = 0.9;     //0.5;
                                lineaGrafica.alpha = valorCorriente/10;
                                corrienteGrafica.gotoAndPlay("i"+valorCorriente.toString());
                                corrienteGrafica.rotationY = 0;
                                if (deltaV<0) {
                                        corrienteGrafica.rotationY = 180;
                                }
                        }

                        corrienteGrafica.visible = true;
                }

                public function desactivarComportamiento3D():void {

                        contenedorGrafico.z = 0;
                        datosComponente[0][4] = contenedorGrafico.z;

                        contenedorGrafico.rotationY = 0;
                        lineaGrafica.scaleX = 1;
                        lineaGrafica.alpha = 1;
                        corrienteGrafica.scaleX = 1;
                        corrienteGrafica.gotoAndStop("i0");

                        inicializaListeners();
                }

        }
}
```

```
package com.fabrica.productor.componentes.fuentesVoltaje
{
	// ************************************************************
	//
	//		Clase:  FuenteVDC (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "Componente2T"
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Clase base
	import com.fabrica.productor.fabricacion.Componente2T;

	public class FuenteVDC extends Componente2T {

		// Tipo de Fuente de Voltaje
		public var i_tipoFuente:String;
		// Diferencia de tension normalizada
		private var dv:Number;
		// Distancia entre terminales del componente
		private var distancia:Number;
		// Angulo entre las terminales 1 y 2
		private var angulo:Number;

		// ------------------------------------------------------------
		//
		//		Seccion de ESPECIFICACION de la fuente de voltaje DC
		//
		// ------------------------------------------------------------

		public function FuenteVDC():void {

			// ------------------------------------------------------------
			//                          Constructor del objeto "FuenteVDC"
			// ------------------------------------------------------------

			// --- Genera dibujo del componente ---
			// Capa 0 : "contenedor"
			contenedorGrafico = new MovieClip();
			addChild(contenedorGrafico);

			// Capa 1 : "linea"
			lineaGrafica = new mc_Linea();
			contenedorGrafico.addChild(lineaGrafica);
```

```
		// Capa 2 : "corriente"
		corrienteGrafica = new mc_Corriente();
		corrienteGrafica.z = 0;
		contenedorGrafico.addChild(corrienteGrafica);

		// Capa 3 : "simbolo"
		simboloGrafico = new mc_VDC();
		contenedorGrafico.addChild(simboloGrafico);

		// --- Inicializa parametros ---
		i_tipoComponente = "V";
		i_tipoFuente = "DC";
		u_nombreComponente = "Fuente de Voltaje DC";
		u_valorComponente = 10;			// 10 volts
		u_unidadesComponente = "volts";

		// --- Permite abrir ventana de edicion ---
		ventanaEdicionPermitida = true;
	}

	public function ventanaInformativa():Array {

		// ------------------------------------------------------------
		//				Ventana informativa del objeto "FuenteVDC"
		// ------------------------------------------------------------
		var infoVentana:Array = new Array();
		infoVentana[0] = u_nombreComponente;
		infoVentana[1] = new Array();
		infoVentana[1] = ["Identificador:", , ];
		infoVentana[2] = new Array();
		infoVentana[2] = ["Valor:",u_valorComponente,u_unidadesComponente];
		infoVentana[3] = new Array();
		infoVentana[3] = ["Fuente:","Voltaje","DC"];
		infoVentana[4] = new Array();
		infoVentana[4] = ["Tipo:","Independiente"];

		return infoVentana;
	}

	// ------------------------------------------------------------
	//
	//		Seccion de EVALUACION de VOLTAJES y CORRIENTES
	//
	// ------------------------------------------------------------

	public function asignarVoltajesNodales():void {
		// Asignacion de voltajes nodales
		Vnodo1 = datosComponente[1][3];
		Vnodo2 = datosComponente[2][3];
		deltaV = Vnodo2 - Vnodo1;
	}

	public function asignarCorrientesRama(corriente:Number):void {
		this.corriente = corriente;
	}
```

```
		// ----------------------------------------------------------
		//
		//			Seccion de ACTIVACION DE COMPORTAMIENTO EN 3D
		//
		// ----------------------------------------------------------

		public function activarComportamiento3D(factorEscala3D:Number,

	escala3D_corriente:Object):void {

			removerListeners();
			// Coloca el componente a una altura igual al punto medio de las
			// coordenadas "z" de cada terminal y aplica el factor de escala
			// para normalizar la altura del componente
			contenedorGrafico.z = factorEscala3D*(Vnodo1 + Vnodo2)/2;
			// Actualiza la coordenada "z" del componente
			datosComponente[0][4] = contenedorGrafico.z;

			// Aplica el factor de escala en 3D para normalizar la diferencia
			// de voltajes entre las terminales del componente
			dv = factorEscala3D*deltaV;
			// Calcula distancia geometrica entre terminales del componente
			distancia = Math.sqrt(dv*dv + 1600);
			// Calcula angulo de la terminal 1 a la 2
			angulo = -(Math.atan2(dv,40)/Math.PI)*180;

			// Rota, escala y orienta el componente de acuerdo a los resultados
			contenedorGrafico.rotationY = angulo;
			lineaGrafica.scaleX = distancia/40;

			// --- CORIIENTE ---
			var valorCorriente:Number;
			if (escala3D_corriente.lineal) {
				// Representacion en ESCALA LINEAL
				valorCorriente = 10;
			} else {
				// Representacion en ESCALA LOGARITMICA
				var corrienteMinima:Number =
escala3D_corriente.corrienteMinima;
				var factor_conversion:Number =
escala3D_corriente.factor_conversion;
				valorCorriente = Math.ceil(factor_conversion*LN_LOG*

Math.log(Math.abs(corriente)/corrienteMinima))+1;
			}

			if (Math.abs(corriente) > 1e-10) {
				corrienteGrafica.rotationY = 180;
				if (corriente<0) {
					corrienteGrafica.rotationY = 0;
				}
				corrienteGrafica.scaleX = distancia/40;
				corrienteGrafica.alpha = 0.9;
				lineaGrafica.alpha = valorCorriente/10;
				corrienteGrafica.gotoAndPlay("i"+valorCorriente.toString());
			}
```

```
                corrienteGrafica.visible = true;
            }

            public function desactivarComportamiento3D():void {
                contenedorGrafico.z = 0;
                datosComponente[0][4] = contenedorGrafico.z;

                contenedorGrafico.rotationY = 0;
                lineaGrafica.scaleX = 1;
                lineaGrafica.alpha = 1;
                corrienteGrafica.scaleX = 1;
                corrienteGrafica.gotoAndStop("i0");

                inicializaListeners();
            }

        }
}
```

```
package com.fabrica.productor.componentes.fuentesVoltaje
{
	// ************************************************************
	//
	//		Clase:  FuenteVAC (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "Componente2T"
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Clase base
	import com.fabrica.productor.fabricacion.Componente2T;

	public class FuenteVAC extends Componente2T {

		// Tipo de Fuente de Voltaje
		public var i_tipoFuente:String;
		// Amplitud
		public var u_amplitudVoltaje:Number;
		private var i_unidadesVoltaje:String;
		// Fase
		public var u_faseVoltaje:Number;
		private var i_unidadesFase:String;

		// ------------------------------------------------------------
		//
		//		Seccion de ESPECIFICACION de la fuente de voltaje AC
		//
		// ------------------------------------------------------------

		public function FuenteVAC():void {

			// ------------------------------------------------------------
			//                          Constructor del objeto "FuenteVAC"
			// ------------------------------------------------------------

			// --- Genera dibujo del componente ---
			// Capa 0 : "contenedor"
			contenedorGrafico = new MovieClip();
			addChild(contenedorGrafico);

			// Capa 1 : "linea"
			lineaGrafica = new mc_Linea();
			contenedorGrafico.addChild(lineaGrafica);
```

```
		// Capa 2 : "corriente"
		corrienteGrafica = new mc_Corriente();
		corrienteGrafica.z = 0;
		contenedorGrafico.addChild(corrienteGrafica);

		// Capa 3 : "simbolo"
		simboloGrafico = new mc_VAC();
		contenedorGrafico.addChild(simboloGrafico);

		// --- Inicializa parametros ---
		i_tipoComponente = "V";
		i_tipoFuente = "AC";
		u_nombreComponente = "Fuente de Voltaje AC";

		u_valorComponente = 10;			// 10 Vp
		u_amplitudVoltaje = u_valorComponente;
		u_unidadesComponente = "Vpico";
		i_unidadesVoltaje = u_unidadesComponente;

		u_faseVoltaje = 0;				// 0 grados
		i_unidadesFase = "Grados";

		// --- Permite abrir ventana de edicion ---
		ventanaEdicionPermitida = true;
	}

	public function ventanaInformativa():Array {

		// ------------------------------------------------------------
		//				Ventana informativa del objeto "FuenteVAC"
		// ------------------------------------------------------------
		var infoVentana:Array = new Array();
		infoVentana[0] = u_nombreComponente;
		infoVentana[1] = new Array();
		infoVentana[1] = ["Identificador:", , ];
		infoVentana[2] = new Array();
		infoVentana[2] = ["Valor:",u_amplitudVoltaje,i_unidadesVoltaje];
		infoVentana[3] = new Array();
		infoVentana[3] = ["Fase:",u_faseVoltaje,i_unidadesFase];
		infoVentana[4] = new Array();
		infoVentana[4] = ["Fuente:","Voltaje","AC"];
		infoVentana[5] = new Array();
		infoVentana[5] = ["Tipo:","Independiente"];

		return infoVentana;
	}

	// ------------------------------------------------------------
	//
	//		Seccion de EVALUACION de VOLTAJES y CORRIENTES
	//
	// ------------------------------------------------------------

	public function asignarVoltajesNodales():void {
		// Asignacion de voltajes nodales
		Vnodo1 = datosComponente[1][3];
```

```
                Vnodo2 = datosComponente[2][3];
                deltaV = Vnodo2 - Vnodo1;
        }

        public function asignarCorrientesRama(corriente:Number):void {
                this.corriente = corriente;
        }

        // ----------------------------------------------------------
        //
        //                  Seccion de ACTIVACION DE COMPORTAMIENTO EN 3D
        //
        // ----------------------------------------------------------

        public function activarComportamiento3D(factorEscala3D:Number,

escala3D_corriente:Object):void {
                // Inicializa componente
                colocarComponenteEspacio3D(factorEscala3D);
                // Activar corriente
                activarCorriente(escala3D_corriente);
        }

        public function colocarComponenteEspacio3D(factorEscala3D:Number):void {

                removerListeners();
                // Coloca el componente a una altura igual al punto medio de las
                // coordenadas "z" de cada terminal y aplica el factor de escala
                // para normalizar la altura del componente
                contenedorGrafico.z = factorEscala3D*(Vnodo1 + Vnodo2)/2;
                // Actualiza la coordenada "z" del componente
                datosComponente[0][4] = contenedorGrafico.z;

                // Aplica el factor de escala en 3D para normalizar la diferencia
                // de voltajes entre las terminales del componente
                dv = factorEscala3D*deltaV;
                // Calcula distancia geometrica entre terminales del componente
                distancia = Math.sqrt(dv*dv + 1600);
                // Calcula angulo de la terminal 1 a la 2
                angulo = -(Math.atan2(dv,40)/Math.PI)*180;

                // Rota, escala y orienta el componente de acuerdo a los resultados
                contenedorGrafico.rotationY = angulo;
                lineaGrafica.scaleX = distancia/40;
        }

        public function activarCorriente(escala3D_corriente:Object):void {

                // --- CORIIENTE ---
                var valorCorriente:Number;
                if (escala3D_corriente.lineal) {
                        // Representacion en ESCALA LINEAL
                        valorCorriente = 10;
                } else {
                        // Representacion en ESCALA LOGARITMICA
```

```
                                var corrienteMinima:Number =
escala3D_corriente.corrienteMinima;
                                var factor_conversion:Number =
escala3D_corriente.factor_conversion;
                                valorCorriente = Math.ceil(factor_conversion*LN_LOG*

Math.log(Math.abs(corriente)/corrienteMinima))+1;
                        }

                        if (Math.abs(corriente) > 1e-10) {
                                corrienteGrafica.rotationY = 180;
                                if (corriente<0) {
                                        corrienteGrafica.rotationY = 0;
                                }
                                corrienteGrafica.scaleX = distancia/40;
                                corrienteGrafica.alpha = 0.9;
                                lineaGrafica.alpha = valorCorriente/10;
                                corrienteGrafica.gotoAndPlay("i"+valorCorriente.toString());
                        }
                }

                public function desactivarComportamiento3D():void {
                        contenedorGrafico.z = 0;
                        datosComponente[0][4] = contenedorGrafico.z;

                        contenedorGrafico.rotationY = 0;
                        lineaGrafica.scaleX = 1;
                        lineaGrafica.alpha = 1;
                        corrienteGrafica.scaleX = 1;
                        corrienteGrafica.gotoAndStop("i0");

                        inicializaListeners();
                }

        }
}
```

```
package com.fabrica.productor.componentes.fuentesCorriente
{
	// ************************************************************
	//
	//		Clase:  FuenteIDC (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "Componente2T"
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Clase base
	import com.fabrica.productor.fabricacion.Componente2T;

	public class FuenteIDC extends Componente2T {

		// Tipo de Fuente de Corriente
		public var i_tipoFuente:String;
		// Diferencia de tension normalizada
		private var dv:Number;
		// Distancia entre terminales del componente
		private var distancia:Number;
		// Angulo entre las terminales 1 y 2
		private var angulo:Number;

		// ------------------------------------------------------------
		//
		//		Seccion de ESPECIFICACION de la fuente de corriente DC
		//
		// ------------------------------------------------------------

		public function FuenteIDC():void {

			// ------------------------------------------------------------
			//                          Constructor del objeto "FuenteIDC"
			// ------------------------------------------------------------

			// --- Genera dibujo del componente ---
			// Capa 0 : "contenedor"
			contenedorGrafico = new MovieClip();
			addChild(contenedorGrafico);

			// Capa 1 : "linea"
			lineaGrafica = new mc_Linea();
			contenedorGrafico.addChild(lineaGrafica);
```

```
            // Capa 2 : "corriente"
            corrienteGrafica = new mc_Corriente();
            corrienteGrafica.z = 0;
            contenedorGrafico.addChild(corrienteGrafica);

            // Capa 3 : "simbolo"
            simboloGrafico = new mc_IDC();
            contenedorGrafico.addChild(simboloGrafico);

            // --- Inicializa parametros ---
            i_tipoComponente = "I";
            i_tipoFuente = "DC";
            u_nombreComponente = "Fuente de Corriente DC";
            u_valorComponente = 10E-3;              // 10 mA
            u_unidadesComponente = "ampere";

            // --- Permite abrir ventana de edicion ---
            ventanaEdicionPermitida = true;
    }

    public function ventanaInformativa():Array {

            // ------------------------------------------------------------
            //                          Ventana informativa del objeto "FuenteIDC"
            // ------------------------------------------------------------
            var infoVentana:Array = new Array();
            infoVentana[0] = u_nombreComponente;
            infoVentana[1] = new Array();
            infoVentana[1] = ["Identificador:", , ];
            infoVentana[2] = new Array();
            infoVentana[2] = ["Valor:",u_valorComponente,u_unidadesComponente];
            infoVentana[3] = new Array();
            infoVentana[3] = ["Fuente:","Corriente","DC"];
            infoVentana[4] = new Array();
            infoVentana[4] = ["Tipo:","Independiente"];

            return infoVentana;
    }

    public function asignarVoltajesNodales():void {
            // Asignacion de voltajes nodales
            Vnodo1 = datosComponente[1][3];
            Vnodo2 = datosComponente[2][3];
            deltaV = Vnodo2 - Vnodo1;

            // Calculo de la corriente
            corriente = u_valorComponente;
    }

    // ------------------------------------------------------------
    //
    //              Seccion de EVALUACION de VOLTAJES y CORRIENTES
    //
    // ------------------------------------------------------------

    public function asignarCorrientesRama(corriente:Number):void {
```

```
                    // --- Dummy ---
                }

                // ----------------------------------------------------------
                //
                //                  Seccion de ACTIVACION DE COMPORTAMIENTO EN 3D
                //
                // ----------------------------------------------------------

                public function activarComportamiento3D(factorEscala3D:Number,

        escala3D_corriente:Object):void {

                        removerListeners();
                        // Coloca el componente a una altura igual al punto medio de las
                        // coordenadas "z" de cada terminal y aplica el factor de escala
                        // para normalizar la altura del componente
                        contenedorGrafico.z = factorEscala3D*(Vnodo1 + Vnodo2)/2;
                        // Actualiza la coordenada "z" del componente
                        datosComponente[0][4] = contenedorGrafico.z;

                        // Aplica el factor de escala en 3D para normalizar la diferencia
                        // de voltajes entre las terminales del componente
                        dv = factorEscala3D*deltaV;
                        // Calcula distancia geometrica entre terminales del componente
                        distancia = Math.sqrt(dv*dv + 1600);
                        // Calcula angulo de la terminal 1 a la 2
                        angulo = -(Math.atan2(dv,40)/Math.PI)*180;

                        // Rota, escala y orienta el componente de acuerdo a los resultados
                        contenedorGrafico.rotationY = angulo;
                        lineaGrafica.scaleX = distancia/40;

                        // --- CORIIENTE ---
                        var valorCorriente:Number;
                        if (escala3D_corriente.lineal) {
                                // Representacion en ESCALA LINEAL
                                valorCorriente = 10;
                        } else {
                                // Representacion en ESCALA LOGARITMICA
                                var corrienteMinima:Number =
escala3D_corriente.corrienteMinima;
                                var factor_conversion:Number =
escala3D_corriente.factor_conversion;
                                valorCorriente = Math.ceil(factor_conversion*LN_LOG*

Math.log(Math.abs(corriente)/corrienteMinima))+1;
                        }

                        if (Math.abs(corriente) > 1e-10) {
                                corrienteGrafica.rotationY = 180;
                                if (corriente<0) {
                                        corrienteGrafica.rotationY = 0;
                                }
                                corrienteGrafica.scaleX = distancia/40;
                                corrienteGrafica.alpha = 0.9;
```

```
                        lineaGrafica.alpha = valorCorriente/10;
                        corrienteGrafica.gotoAndPlay("i"+valorCorriente.toString());
                    }
                }

                public function desactivarComportamiento3D():void {
                    contenedorGrafico.z = 0;
                    datosComponente[0][4] = contenedorGrafico.z;

                    contenedorGrafico.rotationY = 0;
                    lineaGrafica.scaleX = 1;
                    lineaGrafica.alpha = 1;
                    corrienteGrafica.scaleX = 1;
                    corrienteGrafica.gotoAndStop("i0");

                    inicializaListeners();
                }

        }
}
```

```
package com.fabrica.productor.componentes.opamp
{
	// ************************************************************
	//
	//		Clase:  OpAmp (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************
	//
	//		Clase base heredada: "Componente3T"
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Clase base
	import com.fabrica.productor.fabricacion.Componente3T;

	public class OpAmp extends Componente3T {

		public var Av:Number;
		// Diferencia de tension normalizada
		private var dv:Number;
		// Distancia entre terminales del componente
		private var distancia:Number;
		// Angulo entre las terminales 2 y 3
		private var angulo:Number;
		// Inclinacion del compoennte entre las terminales 2 y 3
		private var inclinacion:Number;

		private var V_inversora:Number;
		private var V_noinversora:Number;
		private var Vo:Number;

		// ----------------------------------------------------------
		//
		//		Seccion de ESPECIFICACION del Amplificador Operacional
		//
		// ----------------------------------------------------------

		public function OpAmp():void {

			// ----------------------------------------------------------
			//                          Constructor del objeto "OpAmp"
			// ----------------------------------------------------------

			// --- Genera dibujo del componente ---
			// Capa 0 : "contenedor"
			contenedorGrafico = new MovieClip();
			addChild(contenedorGrafico);
```

```
		// Capa 1 : "linea"
		lineaGrafica = new MovieClip();
		lineaSalida = new mc_OpAmp_LineaSalida();
		lineaEntrada = new mc_OpAmp_LineaEntrada();
		lineaGrafica.addChild(lineaSalida);
		lineaGrafica.addChild(lineaEntrada);
		contenedorGrafico.addChild(lineaGrafica);

		// Capa 2 : "corriente"
		corrienteGrafica = new mc_Corriente2();
		corrienteGrafica.z = 0;
		contenedorGrafico.addChild(corrienteGrafica);
		corrienteGrafica.visible = false;

		// Capa 3 : "simbolo"
		simboloGrafico = new mc_OpAmp();
		contenedorGrafico.addChild(simboloGrafico);

		// --- Inicializa parametros ---
		i_tipoComponente = "A";
		u_nombreComponente = "Amplificador Operacional Ideal";
		u_valorComponente = 1000;		// Ganancia en Lazo Abierto
		u_unidadesComponente = " ";

		// --- Permite abrir ventana de edicion ---
		ventanaEdicionPermitida = true;
	}

	public function ventanaInformativa():Array {

		// ------------------------------------------------------
		//				Ventana informativa del objeto "OpAmp"
		// ------------------------------------------------------
		var infoVentana:Array = new Array();
		infoVentana[0] = u_nombreComponente;
		infoVentana[1] = new Array();
		infoVentana[1] = ["Identificador:", , ];
		infoVentana[2] = new Array();
		infoVentana[2] = ["Av:",u_valorComponente,u_unidadesComponente];

		return infoVentana;
	}

	// ------------------------------------------------------
	//
	//		Seccion de EVALUACION de VOLTAJES y CORRIENTES
	//
	// ------------------------------------------------------

	public function asignarVoltajesNodales():void {
		// Asignacion de voltajes nodales
		Vnodo1 = datosComponente[1][3];	// V_no_inversora
		Vnodo2 = datosComponente[2][3];	// V_inversora
		Vnodo3 = datosComponente[3][3];	// Vo
```

```
                V_noinversora = Vnodo1;
                V_inversora = Vnodo2;
                Vo = Vnodo3;
                deltaV = Vo - V_inversora;
        }

        public function asignarCorrientesRama(corriente:Number):void {
                this.corriente = corriente;
        }

        // ----------------------------------------------------------
        //
        //                      Seccion de ACTIVACION DE COMPORTAMIENTO EN 3D
        //
        // ----------------------------------------------------------

        public function activarComportamiento3D(factorEscala3D:Number,

escala3D_corriente:Object):void {
                // Inicializa componente
                colocarComponenteEspacio3D(factorEscala3D);
                // Activar corriente
                activarCorriente(escala3D_corriente);
        }

        public function colocarComponenteEspacio3D(factorEscala3D:Number):void {

                removerListeners();
                // Coloca el componente a una altura igual al punto medio de las
                // coordenadas "z" de cada terminal y aplica el factor de escala
                // para normalizar la altura del componente
                contenedorGrafico.z = factorEscala3D*(Vnodo3 + Vnodo2)/2;
                // Actualiza la coordenada "z" del componente
                datosComponente[0][4] = contenedorGrafico.z;

                // Aplica el factor de escala en 3D para normalizar la diferencia
                // de voltajes entre las terminales del componente
                dv = factorEscala3D*deltaV;
                // Calcula distancia geometrica entre terminales del componente
                distancia = Math.sqrt(dv*dv + 1600);
                // Calcula angulo de la terminal 1 a la 2
                angulo = (Math.atan2(dv,40)/Math.PI)*180;

                // Rota, escala y orienta el componente de acuerdo a los resultados
                contenedorGrafico.rotationY = -angulo;
                lineaGrafica.scaleX = distancia/40;
        }

        public function activarCorriente(escala3D_corriente:Object):void {

                // --- CORRIENTE ---
                var valorCorriente:Number;
                if (escala3D_corriente.lineal) {
                        // Representacion en ESCALA LINEAL
                        valorCorriente = 10;
                } else {
```

```
                                // Representacion en ESCALA LOGARITMICA
                                var corrienteMinima:Number =
escala3D_corriente.corrienteMinima;
                                var factor_conversion:Number =
escala3D_corriente.factor_conversion;
                                valorCorriente = Math.ceil(factor_conversion*LN_LOG*

Math.log(Math.abs(corriente)/corrienteMinima))+1;
                        }

                        if (Math.abs(corriente) > 1e-10) {
                                corrienteGrafica.rotationY = 0;
                                corrienteGrafica.x = 15; // Para centrar la corriente

                                lineaSalida.alpha = valorCorriente/10;
                                corrienteGrafica.scaleX = distancia/40;
                                corrienteGrafica.gotoAndPlay("i"+valorCorriente.toString());
                                corrienteGrafica.rotationY = 0;

                                if (corriente<0) {
                                        corrienteGrafica.rotationY = 180;
                                }
                        }

                        corrienteGrafica.visible = true;
                }

                public function desactivarComportamiento3D():void {

                        contenedorGrafico.z = 0;
                        datosComponente[0][4] = contenedorGrafico.z;

                        contenedorGrafico.rotationY = 0;
                        lineaGrafica.scaleX = 1;
                        lineaSalida.alpha = 1;
                        corrienteGrafica.scaleX = 1;
                        corrienteGrafica.gotoAndStop("i0");

                        inicializaListeners();
                }

                // ----------------------------------------------------------
                //
                //                                  Seccion de SIMULACION EN AC (3D)
                //
                // ----------------------------------------------------------

                public function activarComportamiento3DComplejo():void {

                        var contenedor_z:Number;

                        removerListeners();
                        // Desactiva corrientes
                        corrienteGrafica.visible = false;
```

```
                datosComponente[0][5] = new Array(); // Ubicacion(t) entre nodos "z" //
[0][5]
                datosComponente[0][6] = new Array(); // Distancia(t) entre nodos          //
[0][6]
                datosComponente[0][7] = new Array(); // Inclinacion(t) entre nodos        //
[0][7]

                // Asignacion del "arreglo de voltajes nodales"
                // a las terminales del componente
                V1_t = datosComponente[1][4]; // [+]
                V2_t = datosComponente[2][4]; // [-]
                V3_t = datosComponente[3][4]; // [Vo]

                for (i in V1_t) {
                        deltaV = V2_t[i] - V3_t[i];
                        // Coloca el componente a una altura igual al punto
                        // medio de las coordenadas "z" de cada terminal
                        contenedor_z = (V2_t[i] + V3_t[i])/2;
                        // Calcula distancia geometrica entre terminales del componente
                        distancia = Math.sqrt(deltaV*deltaV + 1600);
                        // Calcula angulo de la terminal 2 a la 3
                        inclinacion = (Math.atan2(deltaV,40)/Math.PI)*180;

                        datosComponente[0][5].push(contenedor_z);
                        datosComponente[0][6].push(distancia);
                        datosComponente[0][7].push(inclinacion);
                }
        }

        public function asignaPotencial(t:Number):void {

                // ------------------------------------------------------------
                //                      Funcion "onEnterFrame" llamada desde el motor3D_ac
                // ------------------------------------------------------------
                // Rota, escala y orienta el componente
                contenedor_z = datosComponente[0][5][t];
                distancia = datosComponente[0][6][t];
                inclinacion = datosComponente[0][7][t];

                this.z = contenedor_z;
                lineaGrafica.scaleX = distancia/40;
                contenedorGrafico.rotationY = inclinacion;
        }

        public function desactivarComportamiento3DComplejo():void {

                this.z = 0;
                contenedorGrafico.z = 0;
                datosComponente[0][4] = contenedorGrafico.z;

                contenedorGrafico.rotationY = 0;
                lineaGrafica.scaleX = 1;
                lineaSalida.alpha = 1;

                inicializaListeners();
        }
```

```
        }
}
```

```
package com.fabrica.productor.circuitos
{
	// ************************************************************
	//
	//	Clase:	manejadorCircuito (FABRICA)
	//
	//	Patron de Programacion:
	//
	//		"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                mayo /
2010
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Circuitos DC
	import com.fabrica.productor.circuitos.descriptores.dc.circuitoDivisor1;

	// Circuitos AC
	import com.fabrica.productor.circuitos.descriptores.ac.filtroLPActivo1;

	// Fabrica de componentes
	import com.fabrica.productor.fabricacion.FabricarComponente;

	// Procesos de fabricacion de componentes
	import com.fabrica.productor.componentes.resistencia.FabricarResistencia;
	import com.fabrica.productor.componentes.capacitor.FabricarCapacitor;

	import com.fabrica.productor.componentes.alambre.FabricarAlambre1;
	import com.fabrica.productor.componentes.alambre.FabricarAlambre2;
	import com.fabrica.productor.componentes.conexion.FabricarTierra;

	import com.fabrica.productor.componentes.fuentesCorriente.FabricarIDC;
	import com.fabrica.productor.componentes.fuentesVoltaje.FabricarVDC;
	import com.fabrica.productor.componentes.fuentesVoltaje.FabricarVAC;

	import com.fabrica.productor.componentes.opamp.FabricarOpAmp;

	public class manejadorCircuito extends Sprite {

		// --- Nombre del circuito ---
		public var nombre:String;
		// --- Refrencia al area d edicion ---
		public var spEditor:Sprite;
		// --- Parametros comunes ---
		public var parametros:Object;
		// --- Descriptor del circuito ---
		public var circuito:Array;

		public function manejadorCircuito(nombre:String,spEditor:Sprite,

parametros:Object):void {

			// --- Recepcion de datos ---
```

```
		this.nombre = nombre;
		this.spEditor = spEditor;
		this.parametros = parametros;

		// --- Seleccion de circuitos ---
		seleccionaCircuito();
		// --- Generacion del dibujo del circuito ---
		generaDibujoCircuito();
	}

	public function seleccionaCircuito():void {

		// ------------------------------------------------------
		//		Seleccion del circuito
		// ------------------------------------------------------
		var seleccion:Object;
		switch(nombre) {
			case "circuitoDivisor1":
				seleccion = new circuitoDivisor1();
			break;
			case "filtroLPActivo1":
				seleccion = new filtroLPActivo1();
			break;

		}
		circuito = seleccion.descriptorCircuito();
	}

	public function generaDibujoCircuito():void {

		// ------------------------------------------------------
		// Generacion del dibujo del circuito a partir del descriptor
		// ------------------------------------------------------
		var tipoComponente:String;
		var orientacion:String;
		var componente:FabricarComponente;
		var componenteVisual:Object;

		// --- Limpiar spEditor ---
		while (spEditor.numChildren > 0) {
			spEditor.removeChildAt(0);
		}

		// --- Construccion del dibujo del circuito ---
		for (i in circuito) {
			tipoComponente = circuito[i].datosComponente[0][0];
			orientacion = circuito[i].orientacion;
			switch (tipoComponente.substr(0,1)) {
				// --- Resistor -------------------------------------
				case "R":
					componente = new FabricarResistencia();
				break;
				// --- Capacitor ------------------------------------
				case "C":
					componente = new FabricarCapacitor();
				break;
```

```
                        // --- Fuente de Voltaje independiente de DC y AC -----
                        case "V":
                                if (circuito[i].i_tipoFuente == "DC") {
                                        componente = new FabricarVDC();
                                }
                                if (circuito[i].i_tipoFuente == "AC") {
                                        componente = new FabricarVAC();
                                }
                        break;
                        // --- Alambre ----------------------------------------
                        case "W":
                                if (circuito[i].tamanio == "largo") {
                                        componente = new FabricarAlambre1();
                                }
                                if (circuito[i].tamanio == "corto") {
                                        componente = new FabricarAlambre2();
                                }
                        break;
                        // --- Tierra -----------------------------------------
                        case "T":
                                componente = new FabricarTierra();
                        break;
                        // --- Fuente de Corriente Independiente de DC ---------
                        case "I":
                                componente = new FabricarIDC();
                        break;
                        // --- AMPLIFICADOR OERACIONAL "A" --- FVCV ---
                        case "A":       // n1(V+); n2(V-); nc1(Vo)
                                componente = new FabricarOpAmp();
                        break;
                        // --- Default ----------------------------------------
                        default:
                                trace("***********************************************");
                                trace("  CIRCUITO SOLICITADO::  NO
SOPORTADO");
                                trace("***********************************************");
                        break;
                }

                componente.Fabricar(spEditor, parametros, orientacion);
                componenteVisual = spEditor.getChildAt(i);
                componenteVisual.datosComponente =
circuito[i].datosComponente;
                componenteVisual.datosComponente[0][0] =
componenteVisual.name;
                componenteVisual.u_valorComponente =
circuito[i].u_valorComponente;
                // --- Actualiza coordenadas x-y del componente (posicion) ---
                componenteVisual.setComponente();
                // --- Actualiza tipo de Fuente de Voltaje independiente ---
                if (componenteVisual.name.substr(0,1) == "V") {
                        componenteVisual.i_tipoFuente = circuito[i].i_tipoFuente;
                }
                // --- Actualiza Ventana Informativa del componente ---
                componenteVisual.ventanaInformativa();
                // --- Actualiza Ventana de Edicion del componente ---
```

```
componenteVisual.ventanaEdicionComponente.actualizaInformacion();
                        }
                        var obj:Object;

                        // ----------------------------------------------------------
                        //            Para fines de pruebas de funcionamiento del simulador
                        // ----------------------------------------------------------
                        /*trace(" ");
                        trace("manejadorCircuito:: name");
                        for (i=0; i<spEditor.numChildren; i++) {
                                obj = spEditor.getChildAt(i);
                                trace(obj.name+" valor= "+obj.u_valorComponente);
                        }*/
                }
        }
}
```

```
package com.fabrica.productor.circuitos.descriptores.dc
{
	// ************************************************************
	//
	//		Clase:	circuitoDivisor1 (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                septiembre
/ 2010
	// ************************************************************
	//
	//		Descriptor de Circuito
	//
	// ************************************************************

	public class circuitoDivisor1 {

		public function circuitoDivisor1():void {
			// --- Constructor ---
		}

		public function descriptorCircuito():Array {
			//
============================================================
			// Descriptor del circuito:
			//
			//
============================================================

			var circuito:Array = new Array();
			var descriptor:Object;

			descriptor = new Object();
			descriptor.datosComponente = new Array();

			descriptor.datosComponente[0] = new Array();
			descriptor.datosComponente[0][0] = "V0";
			descriptor.datosComponente[0][1] = 218;
			descriptor.datosComponente[0][2] = -40;
			descriptor.datosComponente[0][3] = 0;
			descriptor.datosComponente[0][4] = 0;
			descriptor.datosComponente[1] = [197,-40,-20,0,];
			descriptor.datosComponente[2] = [239,-40,20,0,];
			descriptor.i_tipoFuente = "DC";
			descriptor.u_valorComponente = 10;
			descriptor.orientacion = "N";

			circuito.push(descriptor);

			descriptor = new Object();
			descriptor.datosComponente = new Array();

			descriptor.datosComponente[0] = new Array();
```

```
descriptor.datosComponente[0][0] = "W21";
descriptor.datosComponente[0][1] = 197;
descriptor.datosComponente[0][2] = -40;
descriptor.datosComponente[0][3] = -20;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [176,-40,-40,0];
descriptor.datosComponente[2] = [197,-40,-20,0];
descriptor.tamanio = "corto";
descriptor.u_valorComponente = 0;
descriptor.orientacion = "N";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "W22";
descriptor.datosComponente[0][1] = 177;
descriptor.datosComponente[0][2] = -20;
descriptor.datosComponente[0][3] = -40;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [176,-40,-40,0];
descriptor.datosComponente[2] = [177,-20,-40,0];
descriptor.tamanio = "corto";
descriptor.u_valorComponente = 0;
descriptor.orientacion = "W";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "R3";
descriptor.datosComponente[0][1] = 178;
descriptor.datosComponente[0][2] = 0;
descriptor.datosComponente[0][3] = -40;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [177,-20,-40,0,];
descriptor.datosComponente[2] = [179,20,-40,0,];
descriptor.u_valorComponente = 1000;
descriptor.orientacion = "W";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "W24";
descriptor.datosComponente[0][1] = 180;
descriptor.datosComponente[0][2] = 40;
descriptor.datosComponente[0][3] = -40;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [179,20,-40,0];
```

```
descriptor.datosComponente[2] = [180,40,-40,0];
descriptor.tamanio = "corto";
descriptor.u_valorComponente = 0;
descriptor.orientacion = "W";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "W25";
descriptor.datosComponente[0][1] = 201;
descriptor.datosComponente[0][2] = 40;
descriptor.datosComponente[0][3] = -20;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [180,40,-40,0];
descriptor.datosComponente[2] = [201,40,-20,0];
descriptor.tamanio = "corto";
descriptor.u_valorComponente = 0;
descriptor.orientacion = "N";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "R6";
descriptor.datosComponente[0][1] = 222;
descriptor.datosComponente[0][2] = 40;
descriptor.datosComponente[0][3] = 0;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [201,40,-20,0,];
descriptor.datosComponente[2] = [243,40,20,0,];
descriptor.u_valorComponente = 1000;
descriptor.orientacion = "N";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "T7";
descriptor.datosComponente[0][1] = 239;
descriptor.datosComponente[0][2] = -40;
descriptor.datosComponente[0][3] = 20;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [239,-40,20,0,];
descriptor.u_valorComponente = 0;
descriptor.orientacion = "N";

circuito.push(descriptor);

descriptor = new Object();
```

```
                descriptor.datosComponente = new Array();

                descriptor.datosComponente[0] = new Array();
                descriptor.datosComponente[0][0] = "T8";
                descriptor.datosComponente[0][1] = 243;
                descriptor.datosComponente[0][2] = 40;
                descriptor.datosComponente[0][3] = 20;
                descriptor.datosComponente[0][4] = 0;
                descriptor.datosComponente[1] = [243,40,20,0,];
                descriptor.u_valorComponente = 0;
                descriptor.orientacion = "N";

                circuito.push(descriptor);

                return circuito;
            }

        }
}
```

```
package com.fabrica.productor.circuitos.descriptores.ac
{
	// ************************************************************
	//
	//		Clase:  filtroLPActivo1 (PRODUCTO)
	//
	//		Patron de Programacion:
	//
	//			"Metodo de la Fabrica" (CLIENTE - FABRICA - PRODUCTO)
	//
	//                                                                                  septiembre
/ 2010
	// ************************************************************
	//
	//		Descriptor de Circuito
	//
	// ************************************************************

	public class filtroLPActivo1 {

		public function filtroLPActivo1():void {
			// --- Constructor ---
		}

		public function descriptorCircuito():Array {
			// --- Descriptor del circuito ---

			var circuito:Array = new Array();
			var descriptor:Object;

			descriptor = new Object();
			descriptor.datosComponente = new Array();

			descriptor.datosComponente[0] = new Array();
			descriptor.datosComponente[0][0] = "A0";
			descriptor.datosComponente[0][1] = 222;
			descriptor.datosComponente[0][2] = 40;
			descriptor.datosComponente[0][3] = 0;
			descriptor.datosComponente[0][4] = 0;
			descriptor.datosComponente[1] = [242,220,220,0,0,0];
			descriptor.datosComponente[2] = [200,220,180,0,0,0];
			descriptor.datosComponente[3] = [223,260,200,0,0,0];
			descriptor.u_valorComponente = 1000;
			descriptor.orientacion = "E";

			circuito.push(descriptor);

			descriptor = new Object();
			descriptor.datosComponente = new Array();

			descriptor.datosComponente[0] = new Array();
			descriptor.datosComponente[0][0] = "R1";
			descriptor.datosComponente[0][1] = 199;
			descriptor.datosComponente[0][2] = 0;
			descriptor.datosComponente[0][3] = -20;
			descriptor.datosComponente[0][4] = 0;
```

```
descriptor.datosComponente[1] = [198,-20,-20,0,0,0];
descriptor.datosComponente[2] = [200,20,-20,0,0,0];
descriptor.u_valorComponente = 1000;
descriptor.orientacion = "W";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "R2";
descriptor.datosComponente[0][1] = 201;
descriptor.datosComponente[0][2] = 40;
descriptor.datosComponente[0][3] = -20;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [200,20,-20,0,0,0];
descriptor.datosComponente[2] = [202,60,-20,0,0,0];
descriptor.u_valorComponente = 1000;
descriptor.orientacion = "W";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "V3";
descriptor.datosComponente[0][1] = 219;
descriptor.datosComponente[0][2] = -20;
descriptor.datosComponente[0][3] = 0;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [198,-20,-20,0,0,0];
descriptor.datosComponente[2] = [240,-20,20,0,0,0];
descriptor.i_tipoFuente = "AC";
descriptor.u_valorComponente = 10;
descriptor.orientacion = "N";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "T4";
descriptor.datosComponente[0][1] = 240;
descriptor.datosComponente[0][2] = -20;
descriptor.datosComponente[0][3] = 20;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [240,-20,20,0,0,0];
descriptor.u_valorComponente = 0;
descriptor.orientacion = "N";

circuito.push(descriptor);

descriptor = new Object();
```

```
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "T5";
descriptor.datosComponente[0][1] = 242;
descriptor.datosComponente[0][2] = 20;
descriptor.datosComponente[0][3] = 20;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [242,20,20,0,0,0];
descriptor.u_valorComponente = 0;
descriptor.orientacion = "N";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "C6";
descriptor.datosComponente[0][1] = 180;
descriptor.datosComponente[0][2] = 40;
descriptor.datosComponente[0][3] = -40;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [179,20,-40,0,0,0];
descriptor.datosComponente[2] = [181,60,-40,0,0,0];
descriptor.u_valorComponente = 0.000159;
descriptor.orientacion = "W";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "W27";
descriptor.datosComponente[0][1] = 200;
descriptor.datosComponente[0][2] = 20;
descriptor.datosComponente[0][3] = -20;
descriptor.datosComponente[0][4] = 0;
descriptor.datosComponente[1] = [179,20,-40,0];
descriptor.datosComponente[2] = [200,20,-20,0];
descriptor.tamanio = "corto";
descriptor.u_valorComponente = 0;
descriptor.orientacion = "N";

circuito.push(descriptor);

descriptor = new Object();
descriptor.datosComponente = new Array();

descriptor.datosComponente[0] = new Array();
descriptor.datosComponente[0][0] = "W28";
descriptor.datosComponente[0][1] = 202;
descriptor.datosComponente[0][2] = 60;
descriptor.datosComponente[0][3] = -20;
descriptor.datosComponente[0][4] = 0;
```

```
                descriptor.datosComponente[1] = [181,60,-40,0];
                descriptor.datosComponente[2] = [202,60,-20,0];
                descriptor.tamanio = "corto";
                descriptor.u_valorComponente = 0;
                descriptor.orientacion = "N";

                circuito.push(descriptor);

                descriptor = new Object();
                descriptor.datosComponente = new Array();

                descriptor.datosComponente[0] = new Array();
                descriptor.datosComponente[0][0] = "W29";
                descriptor.datosComponente[0][1] = 223;
                descriptor.datosComponente[0][2] = 60;
                descriptor.datosComponente[0][3] = 0;
                descriptor.datosComponente[0][4] = 0;
                descriptor.datosComponente[1] = [202,60,-20,0];
                descriptor.datosComponente[2] = [223,60,0,0];
                descriptor.tamanio = "corto";
                descriptor.u_valorComponente = 0;
                descriptor.orientacion = "N";

                circuito.push(descriptor);

                return circuito;
            }

        }
}
```

```
package com.bin
{
	// ************************************************************
	//
	//	Clase:	ventanaEdicion
	//
	//	Patron de Programacion:
	//
	//	 Auxiliar de las clases "Componente" y "manejadorCircuito"
	//
	//	Referencias a otras clases:
	//				- "rectangulo"
	//				- "Componente"
	//				- "campoTexto"
	//				- "convertidorPrefijoDecinal"
	//
	//                                                                julio / 2010
	// ************************************************************
	//
	//	ENTRADA:	"componente.ventanaInformativa()"  --> Array:
	//					Arreglo que contiene la informacion del componente
	//					que se desea desplegar y/o editar
	//
	//	SALIDA:			"ventanaEdicion" --> Sprite: Ventana que muestra
	//                                                                datos
informativos/editables
	//                                                                del componente
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Campos de texto para etiquetas
	import flash.text.*;

	// Animaciones
	import fl.transitions.Tween;
	import fl.transitions.TweenEvent;
	import fl.transitions.easing.*;

	import com.fabrica.productor.fabricacion.Componente;

	public class ventanaEdicion extends Sprite {

		// --- Parametros comunes ---
		public var parametros:Object;

		// --- Colores ---
		public var rojo2:Number = 0xcc0000;
		public var naranja2:Number = 0xff9900;
		public var azul2:Number = 0x0066ff;
		public var blanco:Number = 0xffffff;
		public var negro:Number = 0x000000;
```

```
		// --- Elementos de la ventana de edicion ---
		public var target:Sprite;
		public var ventana:Sprite;
		public var ventanaAbierta:Boolean;
		public var cerrarVentana:Boolean;
		// --- Campos de etiquetas ---
		public var campo:campoTexto;
		public var etiqueta_idComponente:TextField;
		// --- Campos de Edicion ---
		public var win_idComponente:TextField; // input
		public var win_nombreComponente:TextField;
		public var win_idComponente_titulo:TextField;
		public var win_valorComponente:TextField;
		public var win_valorComponente_titulo:TextField;
		// --- Animacion de la ventana ---
		private var animacionAbrirVentana:Tween;
		private var animacionCerrarVentana:Tween;
		// --- Sprite para linea de conexion de ventana con componente ---
		public var lineaConexion:Sprite;
		public var ventanaInformativa:rectangulo;
		public var cuadroDesplazamiento:rectangulo;
		public var btnCierre:rectangulo;

		// --- Posicion de componente ---
		public var XposComp:Number;
		public var YposComp:Number;

		// --- Datos del componente ---
		public var componente:Componente;
		public var informacionComponente:Array;

		// --- Analizador del valor del componente ---
		public var ccd:convertidorPrefijosDecimal;

		public function ventanaEdicion(target:DisplayObjectContainer,

componente:Componente,

parametros:Object):void {

				// --- Recepcion de datos ---
				this.target = target;
				this.componente = componente;
				this.parametros = parametros;

				// --- Creacion de la Ventana de Edicion ---
				ventana = new Sprite();
				ventana.name = "ventana";

				// --- Construccion de la Ventana de Edicion ---
				construirVentanaEdicion();
		}

		// ------------------------------------------------------
		//
		//			Seccion de contruccion de la ventana de edicion
```

```
		//
		// ---------------------------------------------------------

		public function construirVentanaEdicion():void {

			// --- Carga informacion del componente para desplegar ---
			informacionComponente = componente.ventanaInformativa();
			// --- Ajuste automatico del "alto de la ventana" ---
			var altoVentana:Number = informacionComponente.length*20;

			// --- Ventana para despliegue de informacion ---
			ventanaInformativa = new rectangulo(200,altoVentana);
			ventanaInformativa.gruesoLinea = 1;
			ventanaInformativa.colorLinea = naranja2;
			ventanaInformativa.alfaLinea = 1;
			ventanaInformativa.colorFondo = negro;
			ventanaInformativa.alfaFondo = 0.6;
			ventanaInformativa.dibujaRectangulo();
			ventana.addChild(ventanaInformativa);

			// --- Encabezado de ventada para desplazamiento ---
			cuadroDesplazamiento = new rectangulo(200,20);
			cuadroDesplazamiento.gruesoLinea = 1;
			cuadroDesplazamiento.colorLinea = naranja2;
			cuadroDesplazamiento.alfaLinea = 1;
			cuadroDesplazamiento.colorFondo = azul2;
			cuadroDesplazamiento.alfaFondo = 1;
			cuadroDesplazamiento.dibujaRectangulo();
			ventana.addChild(cuadroDesplazamiento);
			cuadroDesplazamiento.buttonMode = true;
			cuadroDesplazamiento.addEventListener(MouseEvent.MOUSE_DOWN,
desplazar);
			cuadroDesplazamiento.addEventListener(MouseEvent.MOUSE_UP,
termina_desplazar);

			// --- Boton de cierre de ventana ---
			btnCierre = new rectangulo(10,10);
			btnCierre.x = 185;
			btnCierre.y = 5;
			btnCierre.gruesoLinea = 1;
			btnCierre.colorLinea = rojo2;
			btnCierre.alfaLinea = 1;
			btnCierre.colorFondo = rojo2;
			btnCierre.alfaFondo = 1;
			btnCierre.dibujaRectangulo();
			ventana.addChild(btnCierre);

			// --- Dibujo "X" en el "boton de cierre de ventana" ---
			var dibujoX:Shape = new Shape();
			dibujoX.graphics.lineStyle(2,blanco,1);
			dibujoX.graphics.moveTo(2,2);
			dibujoX.graphics.lineTo(8,8);
			dibujoX.graphics.moveTo(2,8);
			dibujoX.graphics.lineTo(8,2);
```

```
                        btnCierre.addChild(dibujoX);
                        btnCierre.buttonMode = true;
                        btnCierre.addEventListener(MouseEvent.CLICK, cerrar);

                        // ----------------------------------------------------------
                        //                  Ventana Informativa para edicion de parametros
                        //                                              del componente
                        // ----------------------------------------------------------

                        // --- Creacion de Campos de Texto ---
                        campo = new campoTexto(this);

                        // --- Etiqueta Identificador Componente ---
                        etiqueta_idComponente = new TextField();
                        etiqueta_idComponente = campo.estatico(

        componente.u_identificadorComponente,
                                                                naranja2);

                        // ----------------------------------------------------------
                        //                  Datos del componente en la ventana de edicion
                        // ----------------------------------------------------------

                        // + --- Encabezado ---
                        win_nombreComponente = new TextField();
                        win_nombreComponente = campo.estatico(
                                                        informacionComponente[0],
                                                        blanco);
                        ventana.addChild(win_nombreComponente);
                        win_nombreComponente.x = 10;
                        win_nombreComponente.y = 2;

                        // + --- Identificador del componente ---
                                // --- Titulo: Identificador ---
                                win_idComponente_titulo = new TextField();
                                win_idComponente_titulo = campo.estatico(

informacionComponente[1][0],
                                                                        blanco);
                                ventana.addChild(win_idComponente_titulo);
                                win_idComponente_titulo.x = 10;
                                win_idComponente_titulo.y = 25;
                                // --- Editable: id ---
                                win_idComponente = new TextField();

                        // + --- Valor del componente en ventana de edicion ------
                                // --- Titulo: Valor ---
                                win_valorComponente_titulo = new TextField();
                                win_valorComponente_titulo = campo.estatico(

informacionComponente[2][0],
                                                                                blanco);
                                ventana.addChild(win_valorComponente_titulo);
                                win_valorComponente_titulo.x = 10;
                                win_valorComponente_titulo.y = 40;
```

```
                                        // --- Editable: valor ---
                                        // --- Actualiza identificador del componente ---
                                        win_idComponente = new TextField();
                                        win_idComponente = campo.entrada(

componente.u_identificadorComponente,
                                                                        naranja2);
                                        ventana.addChild(win_idComponente);
                                        win_idComponente.x = 100;
                                        win_idComponente.y = 25;
                                        // --- Actualiza valor del componente ---
                                        win_valorComponente = new TextField();
                                        win_valorComponente = campo.entrada(

informacionComponente[2][1],
                                                                            naranja2);
                                        ventana.addChild(win_valorComponente);
                                        win_valorComponente.x = 100;
                                        win_valorComponente.y = 40;

                                        // --- Unidades del campo valor ---
                                        win_unidadesComponente = new TextField();
                                        win_unidadesComponente = campo.estatico(

informacionComponente[2][2],
                                                                                blanco);
                                        ventana.addChild(win_unidadesComponente);
                                        win_unidadesComponente.x = 150;
                                        win_unidadesComponente.y = 40;

                                // + --- Datos informativos del componente en la ventana de edicion ---
                                        if (informacionComponente.length > 3) {
                                                for (var i:uint=3; i<informacionComponente.length; i++) {
                                                        for (var j:uint=0;
j<informacionComponente[i].length; j++) {
                                                                var win:TextField = new TextField();
                                                                win =
campo.estatico(informacionComponente[i][j],

blanco);
                                                                ventana.addChild(win);
                                                                if (j<2) {
                                                                        win.x = 10+j*90;
                                                                } else {
                                                                        win.x = 150;
                                                                }
                                                                win.y = i*18+2;
                                                        }
                                                }
                                        }

                                // --- Linea conexion ventana con componente ---
                                lineaConexion = new Sprite();

                                // --- Habilitacion de cerrarVentana ventana ---
                                cerrarVentana = true;
```

```
                // --- Analizador del valor del componente
                ccd = new convertidorPrefijosDecimal();

        }

        public function update(): void {

                // ---------------------------------------------------------
                // Actualiza parametros del componente cuando se edita algun
                //                          campo de entrada (editable) de la ventana
                // Funcion llamada tambien por la clase "campoTexto"
                // ---------------------------------------------------------

                // ---------------------------------------------------------
                //                          Actualiza IDENTIFICADOR del componente
                // ---------------------------------------------------------
                // --- Actualiza parametros del componente ---
                componente.u_identificadorComponente = win_idComponente.text;

                // --- Actualiza IDENTIFICADOR del componente ---
                target.removeChild(etiqueta_idComponente);
                etiqueta_idComponente = campo.estatico(

componente.u_identificadorComponente,
                                                                naranja2);
                target.addChild(etiqueta_idComponente);
                // --- Actualiza posicion de Campo_idComponente ---
                etiqueta_idComponente.x = XposComp-40;
                etiqueta_idComponente.y = YposComp-10;

                // ---------------------------------------------------------
                //                          Valida y Actualiza VALOR del comnponente
                // ---------------------------------------------------------
                // --- Extraccion del valor numerico del valor del componente ---
                var numero:Number = ccd.analizaDato(win_valorComponente.text);
                if (isNaN(numero)) {
                        win_valorComponente.text = "ERROR!";
                        cerrarVentana = false;
                } else {
                        componente.u_valorComponente = numero;
                        cerrarVentana = true;
                }

                // ---------------------------------------------------------
                //          Actualiza Modelo Nodal y Matriz Nodal del comnponente
                // ---------------------------------------------------------
                //componente.inicializaModeloNodal();
        }

        // ---------------------------------------------------------
        //
        // Seccion de ANIMACION para abrir/cerrar la ventana de edicion
        //
        // ---------------------------------------------------------
```

```
		public function abrir(componente:Componente):void {

			// ---------------------------------------------------------
			//                          Animacion para abrir ventana de edicion
			// ---------------------------------------------------------

			// --- Despliega Ventana de Edicion del componente
			target.addChild(ventana);
			animacionAbrirVentana = new Tween(ventana, "scaleX",

Regular.easeOut, 0.5, 1,
                                                                                0.2, true);
			animacionAbrirVentana = new Tween(ventana, "scaleY",

Regular.easeOut, 0.5, 1,
                                                                                0.2, true);
			ventana.x = componente.x+30;
			ventana.y = componente.y-50;
			target.addChild(etiqueta_idComponente);

			actualizaPosicion(componente);

			ventanaAbierta = true;
		}

		public function cerrar(evt:Event=null):void {

			// ---------------------------------------------------------
			//                          Animacion para cerrar ventana de edicion
			// ---------------------------------------------------------
			update();
			if (cerrarVentana) {
				animacionCerrarVentana = new Tween(ventana, "scaleX",

Regular.easeOut, 1, 0,

0.1, true);
				animacionCerrarVentana = new Tween(ventana, "scaleY",

Regular.easeOut, 1, 0,

0.1, true);
				animacionCerrarVentana.addEventListener(

TweenEvent.MOTION_FINISH,terminaTween);
			}
		}

		public function terminaTween(evnt:Event):void {

			// ---------------------------------------------------------
			// Espera a que la ventana de edicion termine de reducirse
			//                  (animacion) antes de quitarla de la pantalla
			// ---------------------------------------------------------
			target.removeChild(ventana);
			target.removeChild(etiqueta_idComponente);
```

```
                target.removeChild(lineaConexion);
                lineaConexion.graphics.clear();

                ventanaAbierta = false;
                animacionCerrarVentana.removeEventListener(

TweenEvent.MOTION_FINISH,terminaTween);
        }

        // ----------------------------------------------------------
        //
        //              Seccion de desplazamiento de la ventana de edicion
        //
        // ----------------------------------------------------------

        public function desplazar(evt:Event):void {

                // ----------------------------------------------------------
                //                      Desplazamiento de la ventana de edicion
                // ----------------------------------------------------------
                ventana.startDrag();
                target.removeChild(lineaConexion);
                lineaConexion.graphics.clear();
        }

        public function termina_desplazar(evt:Event):void {

                // ----------------------------------------------------------
                //              Termina desplazamiento de la ventana de edicion
                // ----------------------------------------------------------
                ventana.stopDrag();
                lineaConexion.graphics.lineStyle(1,naranja2,1);
                lineaConexion.graphics.moveTo(XposComp, YposComp);
                lineaConexion.graphics.lineTo(ventana.x, ventana.y);
                target.addChild(lineaConexion);
                actualizaPosicion(componente);
        }

        public function actualizaPosicion(componente:Componente):void {

                // ----------------------------------------------------------
                // Actualiza posicion de etiquetas y linea cuando se desplaza
                //                                                  la ventana
                // ----------------------------------------------------------
                XposComp = componente.x;
                YposComp = componente.y;
                // Actualiza posicion de Campo_idComponente
                etiqueta_idComponente.x = XposComp-40;
                etiqueta_idComponente.y = YposComp-10;
                // Trazo de linea de conexion de ventana con componente
                lineaConexion.graphics.clear();
                lineaConexion.graphics.lineStyle(1,naranja2,1);
                lineaConexion.graphics.moveTo(XposComp, YposComp);
                lineaConexion.graphics.lineTo(ventana.x, ventana.y);
                target.addChild(lineaConexion);
        }
```

```
		// ----------------------------------------------------------
		//
		//		Seccion de FUNCIONES utilizadas por la clase externa
		//							"manejadorCircuito"
		//
		// ----------------------------------------------------------

		public function actualizaInformacion():void {

			// ----------------------------------------------------------
			// Funcion utilizada por la clase "manejadorCircuito", cuando
			// se selecciona algun circuito existente, mediante un "boton"
			// en el menu de circuitos. Esta funcion actualiza los VALORES
			// de la ventana de edicion
			// ----------------------------------------------------------

			// --- Actualiza datos del componente ---
			informacionComponente = componente.ventanaInformativa();

			// --- Editable: valor ---
			// --- Actualiza identificador del componente ---
			ventana.removeChild(win_idComponente);
			win_idComponente =
campo.entrada(componente.u_identificadorComponente,
                                                                    naranja2);
			ventana.addChild(win_idComponente);
			win_idComponente.x = 100;
			win_idComponente.y = 25;
			// --- Actualiza valor del componente ---
			ventana.removeChild(win_valorComponente);
			win_valorComponente = campo.entrada(informacionComponente[2][1],
                                                                    naranja2);
			ventana.addChild(win_valorComponente);
			win_valorComponente.x = 100;
			win_valorComponente.y = 40;
		}
	}
}

package com.bin
{
	// ************************************************************
	//
	//	Clase:  rectangulo
	//
	//	Patron de Programacion:
	//
	//				Auxiliar de la clase "ventanaEdicion"
	//
	//                                                              julio / 2010
	// ************************************************************
	//
	//	ENTRADA:	"valorAncho" --> Number: Ancho de la ventana (=50)
	//				"valorAlto" --> Number: Alto de la ventana (=50)
	//				"redondo" --> Boolean: Bordes redondeados (=false)
```

```
	//
	//		SALIDA:			"rectangulo" --> Sprite: Dibujo de un rectangulo
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	public class rectangulo extends Sprite {

		// --- Marco ---
		public var gruesoLinea:uint;
		public var colorLinea:uint;
		public var alfaLinea:Number;

		// --- Fondo ---
		public var colorFondo:uint;
		public var alfaFondo:Number;

		// --- Ancho y alto ---
		public var valorAncho:Number;
		public var valorAlto:Number;

		// --- Esquinas redondeadas ---
		public var redondo:Boolean;
		public var redondeoAncho:Number=8;
		public var redondeoAlto:Number=8;

		public function
rectangulo(valorAncho:Number=50,valorAlto:Number=50,redondo:Boolean=false):void {

			// --- Recepcion de datos para dibujo de la ventana ---
			this.valorAncho = valorAncho;
			this.valorAlto = valorAlto;
			this.redondo = redondo;
		}

		public function dibujaRectangulo():void {

			// --- Dibujo del rectangulo ---
			var bkground:Shape = new Shape();
			bkground.graphics.lineStyle(gruesoLinea,colorLinea,alfaLinea);
			bkground.graphics.beginFill(colorFondo, alfaFondo);
			if (redondo) {
				bkground.graphics.drawRoundRect(0,0,valorAncho,valorAlto,

	redondeoAncho,redondeoAlto);
			} else {
				bkground.graphics.drawRect(0,0,valorAncho,valorAlto);
			}
			bkground.graphics.endFill();
			addChild(bkground);
		}

	}
}
```

```
package com.bin
{
	// ************************************************************
	//
	//		Clase:	campoTexto
	//
	//		Patron de Programacion:
	//
	//					Auxiliar de la clase "ventanaEdicion"
	//
	//                                                                julio / 2010
	// ************************************************************
	//
	//		ENTRADA:	"ventanaEdicion" --> Textos
	//
	//		SALIDA:			textos estatico, dinamico y de entrada
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Campos de texto para etiquetas
	import flash.text.*;

	public class campoTexto {

		public var ventana:Object;
		public var datoEntrada:TextField;

		public function campoTexto(ventanaEdicion:Object):void {
			// --- Registra la referencia a la ventana que lo llama ---
			this.ventana = ventanaEdicion;
		}

		public function estatico(texto:String,color:Number):TextField {

			// ------------------------------------------------------
			//                                          Campo Estatico
			// ------------------------------------------------------
			return dinamico(texto,color);
		}

		public function dinamico(texto:String,color:Number):TextField {

			// ------------------------------------------------------
			//                                          Campo Dinamico
			// ------------------------------------------------------
			var datoSalida:TextField = new TextField();
			datoSalida.autoSize = TextFieldAutoSize.LEFT;
			datoSalida.type = TextFieldType.DYNAMIC;
			datoSalida.text = texto;
			datoSalida = formatoTexto(datoSalida,color);

			return datoSalida;
```

```
		}

		public function entrada(texto:String,color:Number):TextField {

			// ------------------------------------------------------------
			//                                                Campo de Entrada
			// ------------------------------------------------------------
			datoEntrada = new TextField();
			datoEntrada.autoSize = TextFieldAutoSize.LEFT;
			datoEntrada.type = TextFieldType.INPUT;
			datoEntrada.text = texto;
			datoEntrada.addEventListener(KeyboardEvent.KEY_DOWN,
keyDownHandler);
			datoEntrada = formatoTexto(datoEntrada,color);

			return datoEntrada;
		}

		public function formatoTexto(datoTexto:TextField ,color:Number):TextField {

			// ------------------------------------------------------------
			//                                                Formato de Texto
			// ------------------------------------------------------------
			var format:TextFormat = new TextFormat();
			format.font = "_sans";
			format.size = 10;
			format.color = color;
			datoTexto.addEventListener(KeyboardEvent.KEY_DOWN,
keyDownHandler);
			datoTexto.setTextFormat(format);

			return datoTexto;
		}

		public function keyDownHandler(event:KeyboardEvent):void {

			// ------------------------------------------------------------
			//                                        Deteccion de la tecla "ENTER"
			// ------------------------------------------------------------
			// --- Presionar la tecla ENTER (keyCode=13) ---
			var ENTER:Number = 13;
			if (event.keyCode == ENTER) {
				// --- Actualiza parametros del componente ---
				ventana.update();
			}
	}
	}
}
```

```
package com.bin
{
	// ************************************************************
	//
	//	Clase:	convertidorPrefijosDecimal
	//
	//	Patron de Programacion:
	//
	//				Auxiliar de la clase "ventanaEdicion"
	//
	//                                                                julio / 2010
	// ************************************************************
	//
	//	ENTRADA:	"cadena"  --> String: cadena a analizar
	//
	//	SALIDA:			"valorFinal" --> Number
	//
	//	Convierte prefijos del SISTEMA INTERNACIONAL a valores numericos
	//  en notacion decimal (cientifica), para su manejo en el simulador.
	//
	//                                              T --> 1E12
	//                                              G --> 1E9
	//                                              M --> 1E6
	//                                              K --> 1E3
	//                                              m --> 1E-3
	//                                              u --> 1E-6
	//                                              n --> 1E-9
	//                                              p --> 1E-12
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Campos de texto para etiquetas
	import flash.text.*;

	public class convertidorPrefijosDecimal {

		public function convertidorPrefijosDecimal():void {
			// --- Constructor ---
		}

		public function analizaDato(cadena:String):Number {

			var potencia:Number;
			var valor:Number = 0;
			var valorFinal:Number;
			var dato:Array = [];
			var indice:uint = 0;
			var primerCaracter:Boolean = true;

			// Analisis de la cadena escrita por el usuario
			for (var i:uint=0; i<cadena.length; i++) {
				if (cadena.charCodeAt(i) > 47
					&& cadena.charCodeAt(i) < 58) {
					// --- Numero ---
```

```
                        dato.push(cadena.charCodeAt(i));
                        indice++;
                } else if (cadena.charCodeAt(i) == 32) {
                        // --- Espacio ---
                } else if (cadena.charCodeAt(i) == 46) {
                        // --- Punto ---
                        dato.push(cadena.charCodeAt(i));
                } else if (cadena.charCodeAt(i) > 57) {
                        // --- Caracter ---
                        if (primerCaracter) {
                                switch(cadena.charAt(i)) {
                                        case "T":
                                                potencia = 1E12;
                                        break;
                                        case "G":
                                                potencia = 1E9;
                                        break;
                                        case "M":
                                                potencia = 1E6;
                                        break;
                                        case "K":
                                                potencia = 1E3;
                                        break;
                                        case "m":
                                                potencia = 1E-3;
                                        break;
                                        case "u":
                                                potencia = 1E-6;
                                        break;
                                        case "n":
                                                potencia = 1E-9;
                                        break;
                                        case "p":
                                                potencia = 1E-12;
                                        break;
                                        default:

trace("************************************");
                                                trace("--- "+cadena.charAt(i)+" ---");
                                                trace("* CARACTER INVALIDO !!!

        *");

trace("************************************");
                                        }
                                        primerCaracter = false;
                                }
                        }
                        if (i == (cadena.length-1) &&
                                        primerCaracter) {
                                        potencia = 1;
                                        primerCaracter = false;
                        }
                }

                if (potencia != 1) {
                        // ------------------------------------------------
```

```
                        //              Valor expresado en NOTACION CIENTIFICA
                        // ------------------------------------------------
                        // Comprobacion de existencia de punto decimal
                        for (i=0; i<dato.length; i++) {
                                if (cadena.charCodeAt(i) == 46) {
                                        // --- Punto decimal ---
                                        indice--;
                                }
                        }
                        // Obtención del valor numérico del componente
                        for (i=0; i<dato.length; i++) {
                                if (cadena.charCodeAt(i) != 46) {
                                        // --- Punto decimal ---
                                        dato[i] = (dato[i]-48)*Math.pow(10,(indice-1));
                                        indice--;
                                }
                        }

                        // Construccion del valor del componente
                        for (i=0; i<dato.length; i++) {
                                if (cadena.charCodeAt(i) != 46) {
                                        // --- Punto decimal ---
                                        valor += dato[i];
                                }
                        }
                } else {
                        // ------------------------------------------------
                        //              Valor expresado en NOTACION DECIMAL
                        // ------------------------------------------------
                        valor = Number(cadena);
                }
                // Valor final
                valorFinal = valor*potencia;

                return valorFinal;
        }
    }
}
```

```
package com.bin
{
	// ************************************************************
	//
	//		Clase:	botonCircuito
	//
	//		Patron de Programacion:
	//
	//		 Auxiliar de las clases: (circuitos)
	//                                                              - "botonDivisor1"
	//                                                              - "botonFiltroLPActivo1"
	//                                                              - ""
	//                                                              - ""
	//                                                              - ""
	//                                                              - ""
	//                                                              - ""
	//                                                              - ""
	//                                                              - ""
	//
	//                                                                                      septiembre
/ 2010
	// ************************************************************
	//
	//		ENTRADA:	"color"  --> Number: Color del boton
	//                                      "upLogo" --> MovieClip: Logo permanente
	//                                      "overLogo" --> MovieClip: Logo al pasar el mouse
	//
	//		SALIDA:                 "boton" --> SimpleButton
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Rectangulo
	import com.bin.rectangulo;

	public class botonCircuito extends SimpleButton {

		public function botonCircuito(color:Number):void {

			// --- Estado "up" --> "logo up" ---
			upState = new rectangulo(100,100);
			upState.gruesoLinea = 1;
			upState.colorLinea = color;
			upState.alfaLinea = 0.8;
			upState.colorFondo = 0xffffff;
			upState.alfaFondo = 0.1;
			upState.dibujaRectangulo();

			// --- Estado "over" --> "logo over" ---
			overState = new rectangulo(100,100);
			overState.gruesoLinea = 3;
			overState.colorLinea = 0xffffff;
			overState.alfaLinea = 0.8;
			overState.colorFondo = 0xffffff;
```

```
                overState.alfaFondo = 0.3;
                overState.dibujaRectangulo();

                // --- Estado "down" --> "fondo blanco" ---
                downState = new rectangulo(100,100);
                downState.gruesoLinea = 2;
                downState.colorLinea = 0xffffff;
                downState.alfaLinea = 0.2;
                downState.colorFondo = 0xffffff;
                downState.alfaFondo = 1;
                downState.dibujaRectangulo();

                hitTestState = upState;
            }

            public function agregarLogo(upLogo:MovieClip, overLogo:MovieClip):void {

                // --- Logo "up" ---
                upState.addChild(upLogo);
                upLogo.x = 0;
                upLogo.y = 25;

                // --- Logo "over" ---
                overState.addChild(overLogo);
                overLogo.x = 0;
                overLogo.y = 25;
            }
        }
}
```

```
package com.bin
{
	// ************************************************************
	//
	//		Clase:   boton
	//
	//		Patron de Programacion:
	//
	//		 Auxiliar de las clases: (componentes)
	//                                                              - "botonResistencia"
	//                                                              - "botonCapacitor"
	//                                                              - "botonFuenteVoltajeDC"
	//                                                              - "botonFuenteVoltajeAC"
	//                                                              - "botonFuenteCorrienteDC"
	//                                                              - "botonAlambreCorto"
	//                                                              - "botonAlambreLargo"
	//                                                              - "botonTierra"
	//                                                              - "botonOpAmp"
	//
	//                                                                                  junio / 2010
	// ************************************************************
	//
	//		ENTRADA:	"color"  --> Number: Color del boton
	//                                      "upLogo" --> MovieClip: Logo permanente
	//                                      "overLogo" --> MovieClip: Logo al pasar el mouse
	//
	//		SALIDA:                         "boton" --> SimpleButton
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;

	// Rectnagulo
	import com.bin.rectangulo;

	public class boton extends SimpleButton {

		public function boton(color:Number):void {

			// --- Estado "up" --> "logo up" ---
			upState = new rectangulo();
			upState.gruesoLinea = 1;
			upState.colorLinea = color;
			upState.alfaLinea = 0.8;
			upState.colorFondo = 0xffffff;
			upState.alfaFondo = 0.1;
			upState.dibujaRectangulo();

			// --- Estado "over" --> "logo over" ---
			overState = new rectangulo();
			overState.gruesoLinea = 3;
			overState.colorLinea = 0xffffff;
			overState.alfaLinea = 0.8;
			overState.colorFondo = 0xffffff;
			overState.alfaFondo = 0.3;
```

```
                overState.dibujaRectangulo();

                // --- Estado "down" --> "fondo blanco" ---
                downState = new rectangulo();
                downState.gruesoLinea = 2;
                downState.colorLinea = 0xffffff;
                downState.alfaLinea = 0.2;
                downState.colorFondo = 0xffffff;
                downState.alfaFondo = 1;
                downState.dibujaRectangulo();

                hitTestState = upState;
            }

            public function agregarLogo(upLogo:MovieClip, overLogo:MovieClip):void {

                // --- Logo "up" ---
                upState.addChild(upLogo);
                upLogo.x = 25;
                upLogo.y = 25;

                // --- Logo "over" ---
                overState.addChild(overLogo);
                overLogo.x = 25;
                overLogo.y = 25;
            }
        }
}
```

```
package
{
	// ************************************************************
	//
	//		Clase:   Main
	//
	//		Patron de Programacion:
	//
	//                                          Clase constructora del modelo
	//                                    "MVC" (MODELO - VISTA - CONTROLADOR)
	//
	//                                                                                septiembre
/ 2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Modelo MVC
	import Modelo.ModeloCircuito;
	import Controlador.controlador;
	import Vistas.dc.vistasDC;
	import Vistas.ac.vistasAC;

	public class Main extends MovieClip	{

		public function Main() {

			// --- Modelo ---
			var modelo:ModeloCircuito = new ModeloCircuito(this);
			addChild(modelo);

			// --- Vistas ---
			var muestraResultadosDC:vistasDC = new vistasDC(modelo);
			addChild(muestraResultadosDC);
			var muestraResultadosAC:vistasAC = new vistasAC(modelo);
			addChild(muestraResultadosAC);

			// --- Controlador ---
			var cont:controlador = new controlador(modelo,muestraResultadosDC,

muestraResultadosAC);
			addChild(cont);
		}
	}
}
```

```
package Modelo
{
	// ****************************************************************
	//
	//		Clase:	ModeloCircuito
	//
	//		Patron de Programacion:
	//
	//							"MVC" (MODELO - VISTA - CONTROLADOR)
```

```
//
//                                                                                    septiembre /
2010
// ************************************************************

// Flash principal
import flash.display.*;
// Eventos
import flash.events.*;
// Campos de texto para etiquetas
import flash.text.*;

// Manejador de red de circuitos
import com.red.generador.manejadorRed;
// Simuladores
import com.red.simulador.simuladorDC;
import com.red.simulador.simuladorAC;

// Nodos para Vistas
import Vistas.dc.animacion3D.NodoDC;
import Vistas.ac.animacion3D.NodoAC;

// Matematicas
import com.bin.tablaCosenoidal;

//public class Modelo extends EventDispatcher {
public class ModeloCircuito extends Sprite {

        // --- Autorizacion de cambio de estado ---
        public var autorizacionCambiarEstado:Boolean;

        // --- Dibujo y definicion del circuito ---
        public var spEditor:Sprite;
        private var elementosCircuito:Array;
        public var nodosTerminales:Array;
        public var red:manejadorRed;
        public var redNodal:Array;
        public var C:Array;
        public var circuito:Sprite;

        // --- Voltajes ---
        public var voltajesNodales:Array;
        public var voltajesNodalesComplejos:Array;
        private var voltajeMaximo:Number;
        private var voltajeMaximoMagnitud:Number;
        private var escala3D_voltaje:Number = 80;
        // --- Corrientes ---
        public var corrientesRama:Array;
        public var corrientesComponente:Array;
        private var corrienteMaxima:Number;
        private var corrienteMinima:Number;
        private var escala3D_corriente:Object;

        // --- Simuladores ---
        public var simulador_DC:simuladorDC;
        public var simulador_AC:simuladorAC;
        private var frecuenciaCentral:Number;
        private var simulacionValida:Boolean;
        private var tipoSimulacion:String;

        // Lista de resultados de Magnitud y Fase para
        // respuesta en Frecuencia
        public var LD:Generador_Lista_Datos;

        // --- Variables auxiliares ---
        public var parametros:Object;
        public var nodosVisuales:Array;
        public var contenedor:Sprite;
        public var target:Sprite;
        private var componente:Object;
```

```
// --- Mensajes de estado del simulador ---
public var areaMensajesEstado:MovieClip;

// --- Constantes ---
private var LN_LOG:Number = 0.434294;
// --- Matematicas ---
private var tablaCoseno:tablaCosenoidal;

public function ModeloCircuito(contenedor:Sprite):void {

	this.contenedor = contenedor;

	// --- Matematicas ---
	tablaCoseno = new tablaCosenoidal();

	// --- Mensajes iniciales de estado del simulador ---
	generarAreaMensajes();
	muestraMensajeEstado(">>>  EDITANDO ",
                                                        "blanco","alerta");
	muestraMensajeEstado(">>>  EDITANDO ",
                                                        "blanco","normal");

}

public function generarAreaMensajes():void {

	// Creacion de area de mensajes de estado en la parte
	// superior izquierda de la pantalla
	areaMensajesEstado = new mc_AreaMensajesEstado();
	areaMensajesEstado.x = 351;
	areaMensajesEstado.y = 15;
	addChild(areaMensajesEstado);
}

// ------------------------------------------------------------
//
//	Seccion de GENERACION DE RED DEL CIRCUITO y su VALIDACION
//
// ------------------------------------------------------------

public function capturaElementosCircuito(spEditor:Sprite,parametros:Object):void {

	// --- Captura el dibujo del circuito ---
	this.spEditor = spEditor;
	this.parametros = parametros;

	// Guarda referencias a cada elemento del circuito
	// en el arreglo "elementosCircuito"
	elementosCircuito = new Array();
	for (i=0; i<spEditor.numChildren; i++) {
		elementosCircuito.push(spEditor.getChildAt(i));
	}

	// ------------------------------------------------------------
	//		Para fines de pruebas de funcionamiento del simulador
	// ------------------------------------------------------------
	/*trace(" ");
	trace("MMM:: capturaDibujoCircuito");
	for (i in elementosCircuito) {
		trace("MMM:: elementosCircuito["+i+"].name = "+elementosCircuito[i].name);
	}*/

	// --- Incializa autorizacion para cambio del estado
	// de "edicion" a "dc" o "ac"
	autorizacionCambiarEstado = true;

	// --- Validacion del circuito dibujado ---
	validarElementosCircuito();
}
```

```
public function validarElementosCircuito():void {

	// --- Envia el dibujo al manejador de red ---
	red = new manejadorRed(spEditor);

	// ---------------------------------------------------------
	//  Comprueba que el dibujo del circuito cumpla  con
	//  las siguientes restricciones:
	// ---------------------------------------------------------
	if (spEditor.numChildren == 0) {
		trace(" ");
		trace("*****************************************************");
		trace("*                NO SE AUTORIZA CAMBIO DE ESTADO         *");
		trace("*                          PORQUE NO EXISTE CIRCUITO                          *");
		trace("*****************************************************");
		autorizacionCambiarEstado=false;
		muestraMensajeEstado(">>> NO EXISTE CIRCUITO QUE SIMULAR ",
                                                                "naranja","alerta");
	}
	if (autorizacionCambiarEstado && red.ventanaAbierta()) {
		trace(" ");
		trace("*****************************************************");
		trace("*                NO SE AUTORIZA CAMBIO DE ESTADO         *");
		trace("*   PORQUE EXISTEN VENTANAS DE EDICION ABIERTAS   *");
		trace("*****************************************************");
		autorizacionCambiarEstado=false;
		muestraMensajeEstado(">>> CERRAR VENTANAS DE EDICION DE CADA
COMPONENTE ",
                                                                "naranja","alerta");
	}

	if (autorizacionCambiarEstado && !red.existeTierra()) {
		trace(" ");
		trace("*****************************************************");
		trace("*                NO SE AUTORIZA CAMBIO DE ESTADO         *");
		trace("*    PORQUE NO SE HA DEFINIDO EL NODO DE TIERRA   *");
		trace("*****************************************************");
		autorizacionCambiarEstado=false;
		muestraMensajeEstado(">>>  NO SE HA DEFINIDO EL NODO DE TIERRA",
                                                                "naranja","alerta");
	}

	if (autorizacionCambiarEstado && !red.existeCircuito()) {
		trace(" ");
		trace("*****************************************************");
		trace("*                NO SE AUTORIZA CAMBIO DE ESTADO         *");
		trace("*     PORQUE EXISTE UNO O VARIOS ELEMENTOS       *");
		trace("*                DESCONECTADOS DEL CIRCUITO           *");
		trace("*****************************************************");
		autorizacionCambiarEstado=false;
		muestraMensajeEstado(">>>  ELEMENTOS DESCONECTADOS DEL CIRCUITO",
                                                                "naranja","alerta");
	}

	if (autorizacionCambiarEstado && !red.conexionValidaFuentes()) {
		trace(" ");
		trace("*****************************************************");
		trace("*                NO SE AUTORIZA CAMBIO DE ESTADO         *");
		trace("*    PORQUE UNA O VARIAS FUENTES SE ENCUENTRAN    *");
		trace("*                          EN CONEXION NO VALIDA                    *");
		trace("*****************************************************");
		autorizacionCambiarEstado=false;
		muestraMensajeEstado(">>>  FUENTES EN CONEXION NO VALIDA",
                                                                "naranja","alerta");
	}

}

// ---------------------------------------------------------
```

```
//
//                                        Seccion de SIMULACION EN DC
//
// ------------------------------------------------------------

public function simularCircuitoDC():Boolean {

        tipoSimulacion = "DC";
        // ------------------------------------------------------------
        //  Genera lista de componentes del dibujo del circuito
        // ------------------------------------------------------------
        redNodal = red.construyeRed();

        if (!red.compruebaTierraCircuito()) {
                trace(" ");
                trace("*****************************************************");
                trace("*      SE PRODUJO UN ERROR EN LA SIMULACION       *");
                trace("*     EL CIRCUITO NO SE ENCUENTRA CONECTADO       *");
                trace("*    A TIERRA: REVISAR EL CIRCUITO                *")
                trace("*****************************************************");
                autorizacionCambiarEstado=false;
                muestraMensajeEstado("  CIRCUITO NO CONECTADO A TIERRA",
                                                                "naranja","alerta");
                simulacionValida = false;
        } else {
                // ------------------------------------------------------------
                //  Genera la Descripcion Grafica del circuito para generar
                //              "archivos de circuitos" y asociarlos a los "botones del
                //   menu de circuitos"
                // ------------------------------------------------------------
                red.generaDescripcionGraficaCircuito();

                // --- Actualiza mensaje de estado del simulador ---
                muestraMensajeEstado(">>>  SIMULANDO EN DC",
                                                                "blanco","normal");

                // --- Simula el circuito en DC ---
                simulador_DC = new simuladorDC(redNodal);
                // Obtiene voltajes nodales
                voltajesNodales = simulador_DC.obtenerVoltajesNodales();
                // Obtiene corrientes de rama en fuentes de voltaje
                // de DC y alambres
                corrientesRama = simulador_DC.obtenerCorrientesRama();

                // ------------------------------------------------------------
                //              Para fines de pruebas de funcionamiento del simulador
                // ------------------------------------------------------------
                /*trace(" ");
                trace("MMM:: --- redNodal ---");
                for (nodo in redNodal) {
                        trace("redNodal["+nodo+"] id: "+redNodal[nodo].id+" c:
"+redNodal[nodo].c+" valor: "+redNodal[nodo].valor);
                }*/

                /*trace(" ");
                trace("MMM::   --- VOLTAJES NODALES ---");
                for (i in voltajesNodales) {
                        trace("V["+i+"]= "+voltajesNodales[i]);
                }*/

                /*trace(" ");
                trace("MMM::   --- CORRIENTES RAMA ---");
                for (i in corrientesRama) {
                        trace("corrientesRama["+i+"]= "+corrientesRama[i]);
                }*/

                // ------------------------------------------------------------
                //  Comprueba que no existan errores en la simulacion
                //  por las siguientes condiciones:
                // ------------------------------------------------------------
```

```
                                        simulacionValida = true;
                                        for (i in voltajesNodales) {
                                                if (isNaN(voltajesNodales[i]) ||
                                                        !isFinite(voltajesNodales[i])) {
                                                        simulacionValida = false;
                                                        trace(" ");
                                                        trace("*****************************************************");
                                                        trace("*      SE PRODUJO UN ERROR EN LA SIMULACION
*");
                                                        trace("*       UN VOLTAJE NODAL NO FUE CALCULADO
*");
                                                        trace("*    -->          01  REVISAR EL CIRCUITO
*");
                                                        trace("*****************************************************");
                                                        muestraMensajeEstado(" 01::  ERROR EN LA SIMULACION:
REVISAR EL CIRCUITO",
                                                                                                "naranja","alerta");
                                                }
                                        }

                                        for (i in corrientesRama) {
                                                if (isNaN(corrientesRama[i][1]) ||
                                                        !isFinite(corrientesRama[i][1])) {
                                                        simulacionValida = false;
                                                        trace(" ");
                                                        trace("*****************************************************");
                                                        trace("*      SE PRODUJO UN ERROR EN LA SIMULACION
*");
                                                        trace("*     UNA CORRIENTE DE RAMA NO FUE CALCULADA
*");
                                                        trace("*    -->          02  REVISAR EL CIRCUITO
                *");
                                                        trace("*****************************************************");
                                                        muestraMensajeEstado(" 02::  ERROR EN LA SIMULACION:
REVISAR EL CIRCUITO",
                                                                                                "naranja","alerta");
                                                }
                                        }

                                        // --- La simulacion fue exitosa ---
                                        if (simulacionValida) {
                                                // --- Asigna resultados a cada componente del circuito ---
                                                asignarResultadosSimulacionDC();
                                                // --- Crea nodos visuales para representarlos en 3D ---
                                                crearNodosVisualesDC();
                                        }
                                }

                                // --- La simulacion fue exitosa ---
                                return simulacionValida;
                        }

                        public function asignarResultadosSimulacionDC():void {

                                // ------------------------------------------------------------
                                //                      Asignacion de "voltajes nodales" a las terminales
                                //                      correspondientes de cada elemento del circuito y
                                //              calculo de la corriente a traves de el
                                // ------------------------------------------------------------
                                for (i in red.nodosAsignados) {
                                        componente = spEditor.getChildByName(red.nodosAsignados[i][0]);
                                        for (j=1; j<red.nodosAsignados[i].length; j++) {
                                                // Se utiliza signo negativo por convencion de sistema de ejes x,y,z
                                                componente.datosComponente[j][3] = -
voltajesNodales[red.nodosAsignados[i][j]];
                                        }
                                        componente.asignarVoltajesNodales();
                                }

                                // ------------------------------------------------------------
```

```
                    //              Asignacion de "corrientes de rama" a los alambres (W)
                    //                                                  y fuentes de voltaje de DC (V)
                    // ------------------------------------------------------------
                    for (i in corrientesRama) {
                            componente = spEditor.getChildByName(corrientesRama[i][0]);
                            componente.asignarCorrientesRama(corrientesRama[i][1]);
                    }

                    // --- Normalizacion de voltajes y corrientes ---
                    normalizarVoltajes();
                    normalizarCorrientes();

                    var factorEsclala3D_voltaje:Number;
                    if (voltajeMaximo == 0) {
                            factorEsclala3D_voltaje = 1;
                    } else {
                            factorEsclala3D_voltaje = escala3D_voltaje/voltajeMaximo;
                    }

                    // Asignacion de resultados normalizados a cada componente
                    // del circuito para su representacion en 3D
                    for (i in red.nodosAsignados) {
                            componente = spEditor.getChildByName(red.nodosAsignados[i][0]);

componente.activarComportamiento3D(factorEsclala3D_voltaje,escala3D_corriente);
                    }
            }

            public function crearNodosVisualesDC():void {

                    // ------------------------------------------------------
                    //              Generacion de "nodos graficos" que se agregan en la
                    //   representacion en 3D como "puntos" (MovieClip's) en
                    //                          las uniones de los elementos de circuito
                    // ------------------------------------------------------
                    nodosVisuales = new Array();
                    var nodo:NodoDC;
                    var magnitud:Number;
                    var fase:Number;
                    var x_n:Number, y_n:Number;
                    var z_n:Number;
                    var obj:Object;

                    for (i in red.nodosTerminales) {
                            // Asignacion de las coordenadas (x,y,z) para cada "nodo visual"
                            x_n = parametros.puntosRejilla[red.nodosTerminales[i][0]].xg;
                            y_n = parametros.puntosRejilla[red.nodosTerminales[i][0]].yg;
                            if (voltajeMaximo == 0) {
                                    z_n = voltajesNodales[i];
                            } else {
                                    z_n = (voltajesNodales[i]/voltajeMaximo)*escala3D_voltaje;
                            }
                            // --- Formacion del argumento para el "constructor" del nodo ---
                            obj = new Object();
                            obj.i = i;
                            obj.x_n = x_n;
                            obj.y_n = y_n;
                            obj.z_n = z_n;
                            obj.voltajeNodal = voltajesNodales[i];
                            // --- Creacion del nodo visual ---
                            nodo = new NodoDC(obj);
                            nodosVisuales.push(nodo);
                    }
            }

            public function normalizarVoltajes():void {

                    // --- Determinacion del voltaje nodal maximo ---
                    voltajeMaximo = Math.abs(voltajesNodales[0]);
```

```
		for (i=1; i<voltajesNodales.length; i++) {
			if( voltajeMaximo < Math.abs(voltajesNodales[i])) {
				voltajeMaximo = Math.abs(voltajesNodales[i]);
			}
		}
}

public function normalizarCorrientes():void {

		// Asigna los valores de la corriente en cada componente al
		// arreglo "corrientesComponentes" para determinar las corrientes
		// maxima y minima
		corrientesComponentes = new Array();

		for (i=0; i<spEditor.numChildren; i++) {
			componente = spEditor.getChildAt(i);
			if (componente.name.substr(0,1) != "T") {
				corrientesComponentes.push(Math.abs(componente.corriente));
			}
		}

		// Determina la corriente de rama maxima y minima
		corrientesComponentes.sort(Array.NUMERIC);

		for (i=0; i<corrientesComponentes.length; i++) {
			if (Math.abs(corrientesComponentes[i]) <= 1e-9) {
				corrientesComponentes.splice(i,1);
			}
		}

		corrienteMinima = Math.abs(corrientesComponentes[0]);
		corrienteMaxima = Math.abs(corrientesComponentes[corrientesComponentes.length-1]);

		// ------------------------------------------------------------
		//  Calculo de parametros para la ecuacion de conversion
		//  de valores de corriente (i) a una escala lineal (1<x<10)
		//  mediante la funcion logaritmica:
		//
		//			x = convierte_decadas*log(i/corrienteMinima)
		//
		//  con: 1 < x < 10 ; i = valor de la corriente en el componente
		//
		// ------------------------------------------------------------
		escala3D_corriente = new Object();
		// Se establece por default una escala logaritmica para
		// representar las corrientes en cada rama
		escala3D_corriente.lineal = false;
		var numero_decadas:Number;
		var convierte_decadas:Number;
		if (Math.abs(corrienteMinima) > 1e-9) {
			// Caso: "corrienteMaxima > corrienteMinima"
			numero_decadas = Math.ceil(LN_LOG*Math.log(corrienteMaxima/corrienteMinima));
			if (numero_decadas == 0) {
				// Se establece una escala lineal para la representacion
				// de corrientes en el caso: "corrienteMaxima = corrienteMinima"
				escala3D_corriente.lineal = true;
				convierte_decadas = 1;
			} else {
				convierte_decadas = 9/numero_decadas;
			}
		} else {
			// Caso: corrienteMinima = 0
			escala3D_corriente.lineal = true;
			corrienteMinima = 1e-9;
			numero_decadas = 1;
			convierte_decadas = 9/numero_decadas;
		}

		// Cosnstruccion del argumento que se envia a cada componente
		// para que ajuste el ancho de la "flecha" que representara
```

```
		// el valor de la corriente a traves de dicho componente
		escala3D_corriente.factor_conversion = convierte_decadas;
		escala3D_corriente.corrienteMaxima = corrienteMaxima;
		escala3D_corriente.corrienteMinima = corrienteMinima;
}

public function mostrarMensajeDC():void {
		muestraMensajeEstado(" SIMULACION VALIDA EN > DC <",
                                                                "verde","normal");
}

// ----------------------------------------------------------------
//
//                                        Seccion de SIMULACION EN AC
//
// ----------------------------------------------------------------

public function simularCircuitoAC():Boolean {

		tipoSimulacion = "AC";
		// ----------------------------------------------------------------
		//  Genera lista de componentes del dibujo del circuito
		// ----------------------------------------------------------------
		redNodal = red.construyeRed();

		if (!red.compruebaTierraCircuito()) {
			trace(" ");
			trace("*****************************************************");
			trace("*      SE PRODUJO UN ERROR EN LA SIMULACION       *");
			trace("*     EL CIRCUITO NO SE ENCUENTRA CONECTADO       *");
			trace("*    A TIERRA: REVISAR EL CIRCUITO                *");
			trace("*****************************************************");
			autorizacionCambiarEstado=false;
			muestraMensajeEstado("  CIRCUITO NO CONECTADO A TIERRA",
                                                                "naranja","alerta");
			simulacionValida = false;
		} else {
			// ----------------------------------------------------------------
			//  Genera la Descripcion Grafica del circuito para generar
			//             "archivos de circuitos" y asociarlos a los "botones del
			//   menu de circuitos"
			// ----------------------------------------------------------------
			//red.generaDescripcionGraficaCircuito();

			// --- Actualiza mensaje de estado del simulador ---
			muestraMensajeEstado(" SIMULANDO EN AC",
                                                                "verde","normal");

			// --- Simula el circuito en AC ---
			muestraMensajeEstado("  PROCESANDO . . . ",
                                                                "blanco","normal");
			simulador_AC = new simuladorAC(redNodal);

			// Simula el circuito en cada una de las frecuencias
			// generadas en la Lista de Datos "LD"
			simulador_AC.simular_Barrido_Frecuencia(redNodal);
			//simulador_AC.simularAC_Frecuencia_Especifica(redNodal,1);

			// Obtener los valores de los "voltajesNodalesComplejos" a
			// la frecuencia central (logaritmica) del barrido de frecuencia
			voltajesNodalesComplejos = simulador_AC.obtenerVoltajesNodalesComplejos();

			// ----------------------------------------------------------------
			//               Para fines de pruebas de funcionamiento del simulador
			// ----------------------------------------------------------------
			/*trace(" ");
			trace("MMM:: AC   --- VOLTAJES NODALES COMPLEJOS---");
			for (i in voltajesNodalesComplejos) {
				trace("V["+i+"]= "+voltajesNodalesComplejos[i]);
			}*/
```

```
                                        // ---------------------------------------------------------
                                        //  Comprueba que no existan errores en la simulacion
                                        //  por las siguientes condiciones:
                                        // ---------------------------------------------------------
                                        simulacionValida = true;
                                        if (simulador_AC.validarSimulacionAC()) {
                                                simulacionValida = true;
                                        } else {
                                                simulacionValida = false;
                                                trace(" ");
                                                trace("*****************************************************");
                                                trace("*      SE PRODUJO UN ERROR EN LA SIMULACION AC    *");
                                                trace("*       UN VALOR DE GANANCIA NO ES VALIDO        *");
                                                trace("*    -->              REVISAR EL CIRCUITO
*");
                                                trace("*****************************************************");
                                                muestraMensajeEstado(" 03::  ERROR EN LA SIMULACION: REVISAR
EL CIRCUITO",
                                                                                                        "naranja","alerta");
                                        }

                                        // --- Simulacion exitosa ---
                                        if (simulacionValida) {
                                                // --- Recupera valores de la simulacion ---
                                                LD = simulador_AC.LD;
                                                // --- Craer nodos visuales ---
                                                crearNodosVisuales();
                                                // --- Asignar resultados de la simulacion a cada componente ---
                                                asignarResultadosSimulacionAC();
                                        }
                                }

                                // --- Simulacion exitosa ---
                                return simulacionValida;
                        }

                        public function crearNodosVisuales():void {

                                // ---------------------------------------------------------
                                //              Generacion de "nodos visuales" que se agregan en la
                                //   representacion en 3D como "puntos" (MovieClip's) en
                                //                          las uniones de los elementos de circuito
                                // ---------------------------------------------------------
                                nodosVisuales = new Array();
                                var nodo:NodoAC;
                                var magnitud:Number;
                                var fase:Number;
                                var x_n:Number, y_n:Number;
                                var z_n:Array;
                                var obj:Object;

                                // --- Normalizar magnitud de voltajes nodales ---
                                normalizarMagnitudVoltajesComplejos();

                                for (i in red.nodosTerminales) {
                                        // Asignacion de las coordenadas (x,y,z) para cada "nodo visual"
                                        x_n = parametros.puntosRejilla[red.nodosTerminales[i][0]].xg;
                                        y_n = parametros.puntosRejilla[red.nodosTerminales[i][0]].yg;

                                        // Se utiliza signo negativo por convencion de sistema de ejes x,y,z
                                        // --- V(t) = magnitud*Coseno(wt + fase) ---
                                        magnitud = -voltajesNodalesComplejos[i].x;
                                        fase = -voltajesNodalesComplejos[i].y;
                                        if (voltajeMaximoMagnitud != 0) {
                                                magnitud = magnitud*escala3D_voltaje/voltajeMaximoMagnitud;
                                        }
                                        // ---------------------------------------------------------
                                        //  Genera y asigna a cada nodo visual, una tabla con los
                                        //              valores de voltaje nodal defasados de acur=erdo a los
```

```
                                        //  resultados de la simulacion
                                        // ---------------------------------------------------------------
                                        z_n = tablaCoseno.generaCosenoModulado(magnitud,fase);

                                        // --- Formacion del argumento para el "constructor" del nodo ---
                                        obj = new Object();
                                        obj.i = i;
                                        obj.x_n = x_n;
                                        obj.y_n = y_n;
                                        obj.z_n = z_n;          // Array
                                        obj.magnitud = magnitud;
                                        obj.fase = fase;
                                        obj.voltajeNodalPico = voltajesNodalesComplejos[i].x;
                                        // --- Creacion del nodo visua---
                                        nodo = new NodoAC(obj);
                                        nodosVisuales.push(nodo);
                                }
                        }

                        public function asignarResultadosSimulacionAC():void {

                                // ---------------------------------------------------------------
                                //              Asignacion del "arreglo de voltajes nodales" a las terminales
                                //                      correspondientes de cada elemento del circuito
                                //  Estos valores son listas de 400 puntos de una funcion senoidal
                                //  defasada de acuerdo a los resultados de la simulacion
                                // ---------------------------------------------------------------
                                var magnitud:Number;
                                var fase:Number;
                                for (i in red.nodosAsignados) {
                                        componente = spEditor.getChildByName(red.nodosAsignados[i][0]);
                                        for (j=1; j<red.nodosAsignados[i].length; j++) {
                                                // Se utiliza signo negativo por convencion de sistema de ejes x,y,z
                                                // --- V(t) = magnitud*Coseno(wt + fase) ---
                                                componente.datosComponente[j][4] =
nodosVisuales[red.nodosAsignados[i][j]].potenciales;
                                        }
                                        componente.activarComportamiento3DComplejo();
                                }
                        }

                        public function normalizarMagnitudVoltajesComplejos():void {

                                // Determina la magnitud del voltaje nodal maximo
                                voltajeMaximoMagnitud = Math.abs(voltajesNodalesComplejos[0].x);

                                for (i=0; i<voltajesNodalesComplejos.length; i++) {
                                        if( voltajeMaximoMagnitud < Math.abs(voltajesNodalesComplejos[i].x)) {
                                                voltajeMaximoMagnitud = Math.abs(voltajesNodalesComplejos[i].x);
                                        }
                                }
                        }

                        public function mostrarMensajeAC():void {
                                muestraMensajeEstado(" SIMULACION VALIDA EN > AC <",
                                                                                                "verde","normal");
                        }

                        // ---------------------------------------------------------------
                        //
                        //                                                      Seccion de EDICION
                        //
                        // ---------------------------------------------------------------

                        public function muestraMensajeLiberacion():void {
                                muestraMensajeEstado(">>>  EDITANDO ",
                                                                                                "blanco","normal");
                        }

                        // ---------------------------------------------------------------
```

```
		//
		//                                                    Area de MENSAJES DE ESTADO
		//
		// ------------------------------------------------------------------------

		public function muestraMensajeEstado(msg:String,color:String,tipo:String):void {

			// --- Limpia el area de mensajes ---
			while(areaMensajesEstado.numChildren > 0) {
				areaMensajesEstado.removeChildAt(0);
			}

			// --- Seleccion de color del textdel mensaje ---
			var colorTexto:Number;
			switch(color) {
				case "blanco":
					colorTexto = 0xffffff;
				break;
				case "verde":
					colorTexto = 0x00ff99;
				break;
				case "naranja":
					colorTexto = 0xff9900;
				break;
				default:
					trace("ERROR: Color no soportado");
				break;
			}

			// --- Construccion del texto del mensaje ---
			var mensaje:TextField = new TextField();
			mensaje.text = msg;
			mensaje.x=10;
			mensaje.y=3;
			mensaje.autoSize=TextFieldAutoSize.LEFT;
			mensaje.width = 250;
			mensaje.height = 15;

			// --- Formato del Texto del mensaje ---
			var format:TextFormat = new TextFormat();
			format.font = "_sans";
			format.size = 10;
			format.color = colorTexto;
			mensaje.setTextFormat(format);

			// --- Selecciona el tipo de mensaje ---
			switch(tipo) {
				case "alerta":
					areaMensajesEstado.gotoAndPlay(tipo);
				break;
				case "normal":
					areaMensajesEstado.gotoAndPlay(tipo);
				break;
			}

			// --- Despliega mensaje ---
			areaMensajesEstado.addChild(mensaje);
		}
	}

}
```

```
package Controlador
{
	// ****************************************************************
	//
	//		Clase:	controlador
	//
	//		Patron de Programacion:
	//
	//							"MVC" (MODELO - VISTA - CONTROLADOR)
	//
	//												septiembre / 2010
	// ****************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Campos de texto para etiquetas
	import flash.text.*;
	// Botones
	import fl.controls.Button;
	// Animaciones
	import fl.transitions.Tween;
	import fl.transitions.easing.*;

	// Modelo MVC
	import Modelo.ModeloCircuito;
	import Vistas.dc.vistasDC;
	import Vistas.ac.vistasAC;

	// Menus componentes y circuitos
	import com.fabrica.consumidor.registroBotones.CrearBotonesComponentes;
	import com.fabrica.consumidor.registroBotones.CrearBotonesCircuitos;

	public class controlador extends Sprite {

		public var modelo:ModeloCircuito;

		public var elementosCircuito:Array;

		// --- Colores ---
		public var verde:Number = 0x00ff99;
		public var naranja:Number = 0xff6633;
		public var naranja2:Number = 0xff9900;
		public var azul:Number = 0x0099ff;
		public var azul2:Number = 0x0066ff;
		public var amarillo:Number = 0xffff00;
		public var violeta:Number = 0xff00ff;
		public var rojo:Number = 0xff0000;
		public var rojo2:Number = 0xcc0000;
		public var cafe:Number = 0xcc6633;
		public var blanco:Number = 0xffffff;
		public var gris:Number = 0x666666;
		public var negro:Number = 0x000000;

		// --- Editor ---
		public var anchoFondoEditor:Number = 400;
		public var alturaFondoEditor:Number = 400;
		public var spCentro:Sprite;
		public var spEditor:Sprite;
		public var rejilla:Sprite;
		public var mascara:Sprite;

		// --- Parametros compartidos ---
		public var parametros:Object;
		public var separacionEntrePuntosRejilla:Number = 20;
		public var numeroPuntosAnchoRejilla:Number;
		public var numeroPuntosAltoRejilla:Number;
		public var longitudComponente:Number = 40;
		public var puntosRejilla:Array;
```

```
public var posicionInicialComponenteX:Number = -180;
public var posicionInicialComponenteY:Number = -180;

// --- Menus ---
public var spMenuComponentes:Sprite;
private var spMenuCircuitos:Sprite;
private var mcReferencia3D:MovieClip;

// --- Animaciones ---
private var moverVentanaReferencia3D:Tween;
private var moverMenuComponentes:Tween;
private var moverMenuCircuitos:Tween;

// --- Auxiliares ---
public var estado:String;
public var elementos:Array;

public var muestraResultadosDC:vistasDC;
public var muestraResultadosAC:vistasAC;

public function controlador(modelo:ModeloCircuito,
                                                    muestraResultadosDC:vistasDC,
                                                    muestraResultadosAC:vistasAC) {

        this.modelo = modelo;
        this.muestraResultadosDC = muestraResultadosDC;
        this.muestraResultadosAC = muestraResultadosAC;

        // ------------------------------------------------------------
        //  Creacion de botones para el menu superior de la pantalla
        // ------------------------------------------------------------
        btnEdicion = new Button();
        btnEdicion.x = 20;
        btnEdicion.y = 15;
        btnEdicion.label = "EDICION";                   // Maximo 14 caracteres
        addChild(btnEdicion);

        btnSimDC = new Button();
        btnSimDC.x = 130;
        btnSimDC.y = 15;
        btnSimDC.label = "SIMULAR EN DC";   // Maximo 14 caracteres
        addChild(btnSimDC);

        btnSimAC = new Button();
        btnSimAC.x = 240;
        btnSimAC.y = 15;
        btnSimAC.label = "SIMULAR EN AC";    // Maximo 14 caracteres
        addChild(btnSimAC);

        btnSimDC.addEventListener(MouseEvent.CLICK, this.botonSimularDC);
        btnEdicion.addEventListener(MouseEvent.CLICK, this.botonEditar);
        btnSimAC.addEventListener(MouseEvent.CLICK, this.botonSimularAC);

        // ------------------------------------------------------------
        //  Creacion del Editor con su rejilla para colocacion de
        //  componentes
        // ------------------------------------------------------------
        // --- Centro del editor ---
        spCentro = new Sprite();
        spCentro.x = (anchoFondoEditor/2) + 20;
        spCentro.y = (alturaFondoEditor/2) + 60;

        // --- Rejilla Editor ---
        rejilla = new Sprite();
        generaRejilla(rejilla);
        spCentro.addChild(rejilla);

        // --- Editor ---
        spEditor = new Sprite();
        spEditor.graphics.lineStyle(1,blanco,1);
```

```
spEditor.graphics.beginFill(negro,0.5);
spEditor.graphics.drawRect(-(anchoFondoEditor/2+10),
                                                    -(alturaFondoEditor/2+10),
                                                    anchoFondoEditor+20,
                                                    alturaFondoEditor+20);
spEditor.graphics.endFill();
spCentro.addChild(spEditor);

// --- Mascara de edicion aplicada en las simulaciones en 3D ---
mascara = new Sprite();
mascara.graphics.lineStyle();
mascara.graphics.beginFill(0xff0000);
mascara.graphics.drawRect(-(anchoFondoEditor/2+10),
                                                    -(alturaFondoEditor/2+10),
                                                    anchoFondoEditor+21,
                                                    alturaFondoEditor+21);
mascara.graphics.endFill();

// --- Indicador de estado ---
estado = "editando";

// ----------------------------------------------------------
// Parametros comunes a varias Clases y Objetos derivados
// ----------------------------------------------------------
parametros = new Object();
parametros.separacionEntrePuntosRejilla = separacionEntrePuntosRejilla;
parametros.numeroPuntosAnchoRejilla = numeroPuntosAnchoRejilla;
parametros.numeroPuntosAltoRejilla = numeroPuntosAltoRejilla;
parametros.longitudComponente = longitudComponente;
parametros.puntosRejilla = puntosRejilla;
parametros.posicionInicialComponenteX = posicionInicialComponenteX;
parametros.posicionInicialComponenteY = posicionInicialComponenteY;
// ----------------------------------------------------------

// ----------------------------------------------------------
//  Creacion del Menu de Componentes
// ----------------------------------------------------------
spMenuComponentes = new Sprite();
spMenuComponentes.graphics.lineStyle(1,blanco,1);
spMenuComponentes.graphics.beginFill(negro,1);
spMenuComponentes.graphics.drawRect(0,0,300,420);
spMenuComponentes.graphics.endFill();

spMenuComponentes.x = 800;
spMenuComponentes.y = 50;

entradaMenuComponentes();

// ----------------------------------------------------------
// Creacion y colocacion de todos los botones
// correspondientes a los COMPONENTES que se manejan en
// el Menu de Componentes
// ----------------------------------------------------------
var botones:CrearBotonesComponentes = new CrearBotonesComponentes(
                                            spMenuComponentes,
                                              spEditor,parametros);

// ----------------------------------------------------------
//  Creacion del Menu de Circuitos
// ----------------------------------------------------------
spMenuCircuitos = new Sprite();
spMenuCircuitos.graphics.lineStyle(1,blanco,1);
spMenuCircuitos.graphics.beginFill(negro,1);
spMenuCircuitos.graphics.drawRect(0,0,800,200);
spMenuCircuitos.graphics.endFill();

spMenuCircuitos.x = 0;
spMenuCircuitos.y = 600;

entradaMenuCircuitos();
```

```
		// ----------------------------------------------------------
		//  Creacion de la ventana de referencia y controles en 3D
		// ----------------------------------------------------------
		mcReferencia3D = new mc_Referencia_3D();
		mcReferencia3D.x = 732;
		mcReferencia3D.y = 97;

		salidaVentanaReferencia3D();

		// ----------------------------------------------------------
		// Creacion y colocacion de todos los botones
		// correspondientes a los CIRCUITOS que se manejan en
		// el Menu de Circuitos
		// ----------------------------------------------------------
		var botonesCircuitos:CrearBotonesCircuitos = new CrearBotonesCircuitos(
								spMenuCircuitos,
								spEditor,parametros);

		// ----------------------------------------------------------
		//		Colocacion de la ventana de referencia, editor,
		//   menu de componentes y menu de circuitos en capas ordenadas
		// ----------------------------------------------------------
		// --- Capa 1 ---
		addChild(mcReferencia3D);
		// --- Capa 2 ---
		addChild(spMenuComponentes);
		// --- Capa 3 ---
		addChild(spMenuCircuitos);
		// --- Capa 4 ---
		addChild(spCentro);

}

// ----------------------------------------------------------
//
//		Creacion de la Rejilla de referencia pra el Editor
//
// ----------------------------------------------------------

public function generaRejilla(target:Sprite):void {
		// ----------------------------------------------------------
		//     Generador de coordenadas para posicionamiento de
		//   componentes del circuito en la rejilla de dibujo del
		//                                                                  EDITOR
		// ----------------------------------------------------------

		// ----------------------------------------------------------
		// Definicion de arreglo de puntos de referencia
		// Estructura de DATOS:
		// puntosRejilla[numero_Punto] = [fila, columna, x, y, z]
		// ----------------------------------------------------------
		puntosRejilla = new Array();

		// --- Ajuste de tamanio de parametros de la rejilla ---
		numeroPuntosAnchoRejilla = Math.ceil(anchoFondoEditor/separacionEntrePuntosRejilla);
		numeroPuntosAltoRejilla = Math.ceil(alturaFondoEditor/separacionEntrePuntosRejilla);
		mitadNumPuntosXRejilla = numeroPuntosAnchoRejilla/2;
		mitadNumPuntosYRejilla = numeroPuntosAltoRejilla/2;

		// --- Llenado del arreglo de puntos de referencia ---
		var k:Number = 0;
		// --- Coordenadas de DIBUJO ---
		var xg:Number;
		var yg:Number;
		var zg:Number;
		// --- Coordenadas de COLOCACION DE COMPONENTES ---
		var x_coord:Number;
		var y_coord:Number;
		var datos:Object;
```

```
// --- Manejo del Array "puntosRejilla ---
var columna:Number;
var fila:Number;

// ------------------------------------------------------------
// Dibujo de la Rejilla de Referencia para dibujar el circuito
// ------------------------------------------------------------

// --- Centro de la rejilla ---
var long:uint = 6;
target.graphics.lineStyle(1,0xffffff,1);
target.graphics.moveTo(-long,0);
target.graphics.lineTo(long,0);
target.graphics.moveTo(0,-long);
target.graphics.lineTo(0,long);

// --- Circulos en posiciones validas de la rejilla ---
target.graphics.lineStyle(0,0xffffff,0.6);

// i --> RENGLONES (y)
for (var i=-numeroPuntosAltoRejilla/2; i<=numeroPuntosAltoRejilla/2; i++) {
        fila = i + numeroPuntosAltoRejilla/2;
        // j --> COLUMNAS (x)
        for (var j=-numeroPuntosAnchoRejilla/2; j<=numeroPuntosAnchoRejilla/2; j++) {
                columna = j + numeroPuntosAnchoRejilla/2;
                // Coordenadas relativas al centro del editor
                xg = j*separacionEntrePuntosRejilla;
                yg = i*separacionEntrePuntosRejilla;
                zg = 0;
                // Coordenadas relativas a la esquina superior
                //                                      derecha del editor
                x_coord = columna*separacionEntrePuntosRejilla;
                y_coord = fila*separacionEntrePuntosRejilla;
                // ------------------------------------------------------
                //   Dibujo de circulos en cada rejilla, los cuales
                //  indican las posiciones validas para colocacion de
                //                                                      componentes
                // ------------------------------------------------------
                target.graphics.drawCircle(xg,yg,2);

                // ------------------------------------------------------
                // Guardado de DATOS en el arreglo "puntosRejilla"
                // ------------------------------------------------------
                datos = new Object();
                datos.fila = fila;
                datos.columna = columna;
                // Coordenadas relativas a la esquina superior
                // derecha del editor
                datos.x = x_coord;
                datos.y = y_coord;
                datos.z = 0;
                // Coordenadas relativas al centro del editor
                datos.xg = xg;
                datos.yg = yg;
                datos.zg = zg;
                puntosRejilla[k] = datos;

                // ++++++++++++++++++++++++++++++++++++++++++++++++++++++
                // Auxiliar para Proceso de Disenio: Muestra los puntos
                // de la rejilla y etiquetas con sus numeros
                // correspondientes
                // ++++++++++++++++++++++++++++++++++++++++++++++++++++++
                /*var etiqueta:TextField = new TextField();
                var format:TextFormat = new TextFormat();
                etiqueta.text = k;
                //etiqueta.text = xg;
                //etiqueta.text = yg;
                etiqueta.x = xg - 5;
                etiqueta.y = yg - 15;
```

```
                                format.font = "_sans";
                                format.size = 8;
                                format.color = 0xffffff;
                                etiqueta.setTextFormat(format);
                                target.addChild(etiqueta);*/

                                k++;
                        }
                }
                numeroTotalPuntosRejilla = k;
}

// ----------------------------------------------------------
//
//      Controles de entrada y salida del Menu de Componentes
//                      y del Menu de Circuitos
//
// ----------------------------------------------------------

private function entradaVentanaReferencia3D():void {
        moverVentanaReferencia3D = new Tween(mcReferencia3D, "x",
                                        Elastic.easeOut,
                                        mcReferencia3D.x,
                                        732, 3, true);
}

private function salidaVentanaReferencia3D():void {
        moverVentanaReferencia3D = new Tween(mcReferencia3D, "x",
                                        Elastic.easeOut,
                                        mcReferencia3D.x,
                                        1000, 3, true);
}

private function entradaMenuComponentes():void {
        moverMenuComponentes = new Tween(spMenuComponentes, "x",
                                        Elastic.easeOut,
                                        spMenuComponentes.x,
                                        475, 3, true);
}

private function salidaMenuComponentes():void {
        moverMenuComponentes = new Tween(spMenuComponentes, "x",
                                        Elastic.easeOut,
                                        spMenuComponentes.x,
                                        1000, 3, true);
}

private function entradaMenuCircuitos():void {
        moverMenuCircuitos = new Tween(spMenuCircuitos, "y",
                                                                        Elastic.easeOut,
                                                                        spMenuCircuitos.y,
                                                                        500, 3, true);
}

private function salidaMenuCircuitos():void {
        moverMenuCircuitos = new Tween(spMenuCircuitos, "y",
                                                                        Elastic.easeOut,
                                                                        spMenuCircuitos.y,
                                                                        710, 3, true);
}

// ----------------------------------------------------------
//
// Comportamientos de los botones de Edicion, Simular en DC
//                                                      y simular en AC
//
// ----------------------------------------------------------

protected function botonEditar(event:Event):void {
        // ----------------------------------------------------------
```

```
		//		Boton "EDICION "
		// ---------------------------------------------------------
		if (estado != "editando") {

			if (estado == "simulandoDC") {
				// Retorno del estado de SIMULACION EN DC
				// al estado de EDICION
				for (i=0; i<elementosCircuito.length; i++) {
					elementosCircuito[i].desactivarComportamiento3D();
					spEditor.addChild(elementosCircuito[i]);
				}
				// --- Liberar elementos de circuito
				//     del escenario 3D (DC) ---
				muestraResultadosDC.liberarDatos();
			} else {
				// Retorno del estado de SIMULACION EN AC
				// al estado de EDICION
				for (i=0; i<elementosCircuito.length; i++) {
					elementosCircuito[i].desactivarComportamiento3DComplejo();
					spEditor.addChild(elementosCircuito[i]);
				}
				// --- Liberar elementos de circuito
				//     del escenario 3D (AC) ---
				muestraResultadosAC.liberarDatos();
			}

			// --- Mensaje de retorno al estado de "editando" ---
			modelo.muestraMensajeLiberacion();

			// --- Reestablece menus ---
			salidaVentanaReferencia3D()
			entradaMenuComponentes();
			entradaMenuCircuitos();

			// --- Actualiza el estado del simulador ---
			estado = "editando";

			// --- Remueve la mascara del Editor ---
			spCentro.removeChild(mascara);
			spEditor.mask = null;

			// --- Reestablece la rejilla de referenecia ---
			rejilla.visible = true;

		} else {
			// No se permite retornar al estado de EDICION
			// porque ya se encuentra en ese estado
		}
}

protected function botonSimularDC(event:Event):void {
		// ---------------------------------------------------------
		//		Boton "SIMULAR EN DC "
		// ---------------------------------------------------------
		if (estado != "simulandoDC" || estado != "simulandoAC") {
			// Se permite este estado de "simulandoDC"
			// --- Capturar el circuito procedente del Editor
			modelo.capturaElementosCircuito(spEditor,parametros);
			if (modelo.autorizacionCambiarEstado) {
				// --- Captura existosa del circuito ---
				// Guarda referencia a todos los componentes del Editor
				elementosCircuito = new Array();
				for (i=0; i<spEditor.numChildren; i++) {
					elementosCircuito.push(spEditor.getChildAt(i));
				}

				if (modelo.simularCircuitoDC()) {
					// --- Simulacion en DC exitosa ---
					// Procede a mostrar resultados de la simulacion ---
					modelo.mostrarMensajeDC();
```

```
muestraResultadosDC.mostrarDatos(modelo.spEditor,modelo.nodosVisuales);

                                        // --- Quitar menus de componentes y circuitos ---
                                        salidaMenuComponentes();
                                        salidaMenuCircuitos();
                                        entradaVentanaReferencia3D();

                                        // --- Actualiza el estado del simulador ---
                                        estado = "simulandoDC";

                                        // --- Aplica la mascara al Editor ---
                                        // Esta mascara evita que el plano de referencia
                                        // sobrepase el cuadro asignado para mostrar la
                                        // simulacion en 3D
                                        spCentro.addChild(mascara);
                                        spEditor.mask = mascara;

                                        // --- Quitar la rejilla de referencia del Editor ---
                                        rejilla.visible = false;
                                }
                        } else {
                                // Si el simulador se encuentra en el estado "simulandoDC"
                                // no puede cambiar al estado "simulandoAC" o "simulandoDC".
                                // Solo podra cambiar al estado de "editando"
                        }
                }
        }


        protected function botonSimularAC(event:Event):void {
                // ------------------------------------------------------------
                //              Boton "SIMULAR EN AC "
                // ------------------------------------------------------------
                if (estado != "simulandoDC" || estado != "simulandoAC") {
                        // Se permite este estado de "simulandoAC"
                        // --- Capturar el circuito procedente del Editor
                        modelo.capturaElementosCircuito(spEditor,parametros);
                        if (modelo.autorizacionCambiarEstado) {
                                // --- Captura existosa del circuito ---
                                // Guarda referencia a todos los componentes del Editor
                                elementosCircuito = new Array();
                                for (i=0; i<spEditor.numChildren; i++) {
                                        elementosCircuito.push(spEditor.getChildAt(i));
                                }

                                if (modelo.simularCircuitoAC()) {
                                        // --- Simulacion en AC exitosa ---
                                        // Procede a mostrar resultados de la simulacion ---
                                        modelo.mostrarMensajeAC();
                                muestraResultadosAC.mostrarRespuestaFrecuencia(modelo.LD,modelo);
                                muestraResultadosAC.mostrarDatos(modelo.spEditor,modelo.nodosVisuales);

                                        // --- Quitar menus de componentes y circuitos ---
                                        salidaMenuComponentes();
                                        salidaMenuCircuitos();
                                        entradaVentanaReferencia3D();

                                        // --- Actualiza el estado del simulador ---
                                        estado = "simulandoAC";

                                        // --- Aplica la mascara al Editor ---
                                        // Esta mascara evita que el plano de referencia
                                        // sobrepase el cuadro asignado para mostrar la
                                        // simulacion en 3D
                                        spCentro.addChild(mascara);
                                        spEditor.mask = mascara;

                                        // --- Quitar la rejilla de referencia del Editor ---
                                        rejilla.visible = false;
```

```
                                        }
                                } else {
                                        // Si el simulador se encuentra en el estado "simulandoAC"
                                        // no puede cambiar al estado "simulandoDC" o "simulandoAC".
                                        // Solo podra cambiar al estado de "editando"
                                }
                        }
                }
        }
}


package com.bin
{
        // ************************************************************
        //
        //      Clase:   tablaCosenoidal
        //
        //      Patron de Programacion:
        //
        //       Auxiliar de la clase: "modeloCircuito"
        //
        //                                                              septiembre / 2010
        // ************************************************************
        //
        //      TABLA:                  "coseno" ---> Array: 62 muestras de la funcion
        //                                      coseno espaciadas un angulo de 0.1
        //                                              radianes con amplitud 1.0 y fase 0
        //                                      Formato:
        //                                              coseno[i][0] = angulo en radianes
        //                                              coseno[i][1] = valor de la funcion
        //
        //      ENTRADA:        "amplitud"  --> Number: Amplitud del coseno
        //                              "fase"      --> Number: Fase del coseno
        //
        //      SALIDA:                 "cosenoModulado" --> Array: 62 muestras de la
        //                                              funcion coseno espaciadas
        //                                              un angulo de 0.1 radianes
        //                                              modulados con "amplitud" y
        //                                              "fase"
        //
        // ************************************************************

        public class tablaCosenoidal {

                // --- Tabla cosenoidal ---
                public var coseno:Array;
                // --- Incremento del angulo en radianes ---
                public var deltaAngulo:Number = 0.1;
                // --- Numero de muestras de la Tabla cosenoidal ---
                var numMuestras:Number;

                public function tablaCosenoidal():void {

                        // --- Constructor ---
                        generaTablaCosenoidal();
                }
```

```
		public function generaTablaCosenoidal():void {

			// ---------------------------------------------------------
			// Generacion de la tabla cosenoidal con amplitud 1.0 y fase 0
			// ---------------------------------------------------------
			var cos:Number;
			var angulo:Number = 0;

			coseno = new Array();
			// --- 62 muestras ---
			numMuestras = Math.round(2*Math.PI/deltaAngulo)-1;

			for (i=0; i<numMuestras; i++) {
				cos = Math.round(100*Math.cos(angulo))/100;
				// --- Guardado de las muestras: angulo y valor de la funcion ---
				coseno[i] = [angulo, cos];
				angulo = Math.round(10*(angulo+deltaAngulo))/10;
			}
		}

		public function generaCosenoModulado(amplitud:Number,fase:Number):Array {

			// ---------------------------------------------------------
			//          Generacion del coseno modulado con "amplitud" y "fase"
			// ---------------------------------------------------------
			var cosenoModulado:Array = new Array();
			var faseIndiceInicial:Number;
			var faseIndice:Number;

			// --- Generacion del barrido defasado ---
			if (fase >= 0) {
				faseIndiceInicial = Math.round(numMuestras*fase/360);
			} else {
				faseIndiceInicial = numMuestras -
				Math.round(numMuestras*Math.abs(fase)/360);
			}

			// --- Construccion del coseno modulado ---
			for (i in coseno) {
				faseIndice = faseIndiceInicial + i;
				if (faseIndice < coseno.length) {
					cosenoModulado[i] = Math.round(
					amplitud*coseno[faseIndice][1]);
				} else {
					cosenoModulado[i] = Math.round(
							amplitud*coseno[faseIndice -
numMuestras][1]);
				}
			}

			return cosenoModulado
		}

	}
}
```

```
package com.red.generador
{
	// ****************************************************************
	//
	//		Clase:	manejadorRed
	//
	//		Patron de Programacion:
	//
	//					Auxiliar de la clase "ModeloCircuito"
	//
	//													abril
/ 2010
	// ****************************************************************
	//
	//		ENTRADA:
	//		  - redDibujo --> Sprite: Dibujo del circuito a simular
	//
	//		SALIDA:
	//			- Validacion del circuito:
	//			  * existeCircuito() --> Existe circuito?
	//			  * ventanaAbierta() --> Existen ventanas de edicion abiertas?
	//			  * existeTierra() --> Existe nodo de tierra?
	//			  * compruebaTierraCircuito() --> Comprueba que el circuito
	//												tenga
conectado al menos
	//												una terminal
de "tierra"
	//			  * conexionValidaFuentes() --> Busca Fuentes de VOLTAJE en
	//												CORTO
CIRCUITO o
	//												Fuentes de
CORRIENTE en
	//												CIRCUITO
ABIERTO
	//			  * construyeRed() --> Genera un archivo descriptivo de
	//									del circuito con un formato similar
	//									a "PSPICE"
	//			  * generaDescripcionGraficaCircuito() --> Genera el archivo
	//
	impreso de la descripcion
	//													del
circuito dibujado en
	//													el
editor
	//
	// ****************************************************************
	import flash.display.Sprite;

	public class manejadorRed {

		public var redDibujo:Sprite;

		public var redCircuito:Array;
		public var listaTerminales:Array;
		public var listaComponentes:Array;
		public var listaAlambres:Array;
		public var nodosAlambres:Array;
		public var trayectos:Array;
		public var nodosTerminales:Array;
		public var nodosAsignados:Array;
		public var C:Array;

		public function manejadorRed(redDibujo:Sprite):void {

			this.redDibujo = redDibujo;
			redCircuito = new Array();
			listaTerminales = new Array();
			listaComponentes = new Array();
			listaAlambres = new Array();
			nodosAlambres = new Array();
```

```
		trayectos = new Array();
		nodosTerminales = new Array();
		nodosAsignados = new Array();
		C = new Array();
}

// ----------------------------------------------------------
//
//                                  Seccion de Validacion del circuito
//
// ----------------------------------------------------------

public function existeCircuito():Boolean {

		genera_listaComponentes_redCircuito();

		// Busca elementos que no esten conectados al circuito
		var existe:Boolean = true;
		for (k=0; k<listaComponentes.length-1; k++) {
			if (listaComponentes[k].length < 2) {
				for (i=k+1; i<listaComponentes.length; i++) {
					if (listaComponentes[i].length < 2 &&
						listaComponentes[k][0] == listaComponentes[i][0]) {
						existe = false;
						break;
					}
				}
			}
		}
		return existe;
}

public function ventanaAbierta():Boolean {

		var abierta:Boolean = false;
		// ----------------------------------------------------------------
		//  Comprueba que el dibujo del circuito contenga solamente
		//  elementos de circuito y no "ventanas abiertas"
		// ----------------------------------------------------------------
		for (i=0; i < redDibujo.numChildren; i++) {
			if (redDibujo.getChildAt(i).name == "ventana") {
				trace(" ");
				trace("*****************************************************");
				trace("*                                          --- ERROR ---
 *");
				trace("*    El circuito contiene ventanas abiertas         *");
				trace("*****************************************************");
				abierta = true;
				break;
			}
		}
		return abierta;
}

public function existeTierra():Boolean {

		genera_listaTerminales_redCircuito();
		// Busca unicamente Tierra en el arreglo redCircuito
		var existe:Boolean = false;
		for (k in redCircuito) {
			if (redCircuito[k][0].substr(0,1)=="T") {
				existe = true;
				break;
			}
		}
		return existe;
}

public function conexionValidaFuentes():Boolean {
```

```
		// Busca Fuentes de VOLTAJE en CORTO CIRCUITO o
		// Fuentes de CORRIENTE en CIRCUITO ABIERTO
		var conexionValida:Boolean = true;
		for (k=0; k<listaComponentes.length; k++) {
			if ((listaComponentes[k].length < 2) &&
				(listaComponentes[k][0].substr(0,1) == "I")) {
				// Fuentes de CORRIENTE en CIRCUITO ABIERTO
				conexionValida = false;
				break;
			}
		}
		return conexionValida;
}

public function	genera_listaComponentes_redCircuito():void {

		for (i in listaTerminales) {
			listaComponentes[i] = new Array();
			for (j in redCircuito) {
				if( redCircuito[j][1] == listaTerminales[i] ||
					redCircuito[j][2] == listaTerminales[i]) {
					listaComponentes[i].push(redCircuito[j][0]);
				}
			}
		}
}

public function	genera_listaTerminales_redCircuito():void {

		var componente:Object;
		var informacionComponente:Array;

		for (i=0; i < redDibujo.numChildren; i++) {
			componente = redDibujo.getChildAt(i);
			informacionComponente = new Array();
			for (j=0; j<componente.datosComponente.length; j++) {
				informacionComponente.push(componente.datosComponente[j][0]);
				// [0][0] --> identificador del componente
				// [1][0] --> terminal 1
				// [2][0] --> terminal 2
				// [3][0] --> terminal 3
				if (j) {
					listaTerminales.push(componente.datosComponente[j][0]);
				}
			}
			redCircuito.push(informacionComponente);
		}

		// --- Ordena y reduce conexiones ---
		ordenaLista(listaTerminales);
}

public function ordenaLista(a:Array):void {

		// --- Ordenamiento ascendente del arreglo --------
		a.sort(Array.NUMERIC);

		// --- Eliminacion de valores repetidos -----------
		for (var j=0; j<a.length; j++) {
			for (var i=j+1; i<a.length; i++) {
				if (a[j]==a[i]) {
					a.splice(i,1);
					i-=1;
				}
			}
		}
}

// ------------------------------------------------------------
//
```

```
// Seccion de construccion de la red descriptiva del circuito
//
// ----------------------------------------------------------

public function construyeRed():Array {

	generaRedNodal();
	generaRedCircuito();

	return C;
}

public function generaRedNodal():void {

	var j:uint;
	var a:uint = 0;
	nodosTerminales[a] = new Array();

	// --- Genera una lista de "terminales" de Tierra ---
	for (k in redCircuito) {
		if (redCircuito[k][0].substr(0,1)=="T") {
			nodosTerminales[0].push(redCircuito[k][1]);
		}
	}
	a++;

	// Del arreglo "listaTerminales" elimina "terminales"
	// ya incluidas en el arreglo "nodosTerminales"
	var terminalNueva:Boolean = true;
	for (i in listaTerminales) {
		for (k=0; k<nodosTerminales[0].length; k++) {
			if (listaTerminales[i] == nodosTerminales[0][k]) {
				terminalNueva = false;
			}
		}
		if (terminalNueva) {
			nodosTerminales[a] = new Array();
			nodosTerminales[a].push(listaTerminales[i]);
			a++;
		}
		terminalNueva = true;
	}
}

public function generaRedCircuito():void {

	// ----------------------------------------------------------
	// Asigna "nodos" a cada "terminal" de cada "componente" de la
	// red que describe al circuito
	// ----------------------------------------------------------
	for (i in redCircuito) {
		nodosAsignados[i] = new Array();
		nodosAsignados[i].push(redCircuito[i][0]);
		for (j=1; j<redCircuito[i].length; j++) {
			for (var nodo in nodosTerminales) {
				for (k in nodosTerminales[nodo]) {
					if (redCircuito[i][j] == nodosTerminales[nodo][k]) {
						nodosAsignados[i].push(nodo);
					}
				}
			}
		}
	}

	// ----------------------------------------------------------
	// Genera el arreglo "C" con el "formato adecuado" para que lo
	// pudea interpretar el "simulador":
	//                                                              -
```

Analisis_Nodal_Modificado_Complejo

```
                    //                                                  -
Soulcion_Sistema_Ecuaciones_Complejo
                    // ----------------------------------------------------------
                    var obj:Object;

                    for (nodo in nodosAsignados) {
                              obj = new Object();
                              if (nodosAsignados[nodo][0].substr(0,1) != "T") {
                                        // --- Identificador del componente ---
                                        obj.id = nodosAsignados[nodo][0];
                                        // --- Nodos de conexion del componente ---
                                        obj.c = new Array();
                                        for (i=1; i<nodosAsignados[nodo].length; i++) {
                                                  obj.c.push(nodosAsignados[nodo][i]);
                                        }

                                        // --- Valor del componente ---
                                        obj.valor = redDibujo.getChildByName(obj.id).u_valorComponente;
                                        // --- Tipo de Fuente de Voltaje o Corriente Independiente ---
                                        //                                                  "DC" o "AC"
                                        if (nodosAsignados[nodo][0].substr(0,1) == "V" ||
                                                  nodosAsignados[nodo][0].substr(0,1) == "I") {
                                                  obj.tipo = redDibujo.getChildByName(obj.id).i_tipoFuente;
                                        }
                                        C.push(obj);
                              }
                    }

                    // ----------------------------------------------------------
                    //              Para fines de pruebas de funcionamiento del simulador
                    // ----------------------------------------------------------
                    trace(" ");
                    trace(" --- C ---");
                    for (nodo in C) {
                              trace("C["+nodo+"] id: "+C[nodo].id+" c: "+C[nodo].c+
                                        " valor: "+C[nodo].valor);
                    }
          }

          public function compruebaTierraCircuito():Boolean {

                    var conexionTierra:Boolean = false;
                    // ----------------------------------------------------------
                    // Comprueba que el circuito tenga conectado al menos una
                    // terminal de "tierra"
                    // ----------------------------------------------------------
                    for (nodo in C) {
                              for (j=0; j<C[nodo].c.length; j++) {
                                        if (C[nodo].c[j] == 0) {
                                                  conexionTierra = true;
                                                  break;
                                        }
                              }
                    }

                    if (!conexionTierra) {
                              trace(" ");
                              trace("*******************************************************");
                              trace("*                                        --- ERROR ---
 *");
                              trace("*    El circuito no tiene conexion a tierra           *");
                              trace("*******************************************************");
                    }

                    return conexionTierra;
          }

          // ----------------------------------------------------------
          //
          // Seccion de impresion del "descriptor" del circuito generado
```

```
//              automaticamente, a partir del dibujo realizado por el
//                                                                                  usuario
//
// ------------------------------------------------------------

public function generaDescripcionGraficaCircuito():void {

        // ------------------------------------------------------------
        //                  CONSTRUCCION del descritor del circuito
        //                      a partir del dibujo del circuito
        // ------------------------------------------------------------
        var circuito:Array = new Array();
        var componente:Object;
        var descriptor:Object;
        var identificador:String;

        // --- Formato de datos del componente ---
        for (i=0; i < redDibujo.numChildren; i++) {
                descriptor = new Object();
                descriptor.datosComponente = new Array();
                componente = redDibujo.getChildAt(i);
                // --- Identificador ---
                identificador = componente.name.substr(0,1);
                for (j=0; j<componente.datosComponente.length; j++ ) {
                        // datosComponente[0] --> Identificador y coordenadas de
                        //
posicion del componente
                        // datosComponente[1] --> Coordenadas Terminal 1
                        // datosComponente[2] --> Coordenadas Terminal 2 (si existe)
                        // datosComponente[3] --> Coordenadas Terminal 3 (si existe)
                        descriptor.datosComponente.push(componente.datosComponente[j]);
                }
                descriptor.u_valorComponente = componente.u_valorComponente;
                descriptor.orientacion= componente.orientacion;

                if (identificador == "V") {
                        // --- Selecciona tipo de fuente de voltaje ---
                        descriptor.i_tipoFuente = componente.i_tipoFuente;
                }

                if (identificador == "W") {
                        // Selecciona "alambre":
                        //                      - largo (40 pixeles)
                        //                      - corto (20 pixeles)
                        if((Math.abs(componente.datosComponente[1][1]-
                                                componente.datosComponente[2][1])==40)
||
                          (Math.abs(componente.datosComponente[1][2]-
                                                componente.datosComponente[2][2])==40))
{
                                descriptor.tamanio = "largo";
                        } else {
                                descriptor.tamanio = "corto";
                        }
                }

                circuito.push(descriptor);
        }

        // ------------------------------------------------------------
        //                          IMPRESION del descritor del circuito
        //                            a partir del dibujo del circuito
        // ------------------------------------------------------------
        trace(" ");
        trace("// ============================================================");
        trace("// Descriptor del circuito:");
        trace("//");
        trace("// ============================================================");
        trace(" ");
        trace("var circuito:Array = new Array();");
```

```
                    trace("var descriptor:Object;");
                    for (i in circuito) {
                        trace(" ");
                        trace("descriptor = new Object();");
                        trace("descriptor.datosComponente = new Array();");
                        trace(" ");
                        for (j in circuito[i].datosComponente) {
                            if (j == 0) {
                                trace("descriptor.datosComponente[0] = new Array();");
                                for (k=0; k<=4; k++) {
                                //for (k in circuito[i].datosComponente[0]) {
                                    if (k == 0) {
                                        trace("descriptor.datosComponente[0][0] =
\""+

circuito[i].datosComponente[0][0]+"\";");
                                    } else {

            trace("descriptor.datosComponente["+0+"]["+k+
                                                "] =
"+circuito[i].datosComponente[0][k]+";");
                                    }
                                }
                            } else {
                                trace("descriptor.datosComponente["+j+"] = ["+
                                    circuito[i].datosComponente[j]+"];");
                            }
                        }

                        if (circuito[i].datosComponente[0][0].substr(0,1) == "V") {
                            trace("descriptor.i_tipoFuente = \""+
                                circuito[i].i_tipoFuente+"\";");
                        }

                        if (circuito[i].datosComponente[0][0].substr(0,1) == "W") {
                            trace("descriptor.tamanio = \""+circuito[i].tamanio+"\";");
                        }

                        trace("descriptor.u_valorComponente = "+
                            circuito[i].u_valorComponente+";");
                        trace("descriptor.orientacion = "+"\""+
                            circuito[i].orientacion+"\";");
                        trace(" ");
                        trace("circuito.push(descriptor);");

                    }

                    trace(" ");
                    trace("return circuito;");
                    trace(" ");
                    trace(" ============================================================");
                    trace(" ---------- FIN: Descriptor del circuito ----------");
                    trace(" ============================================================");
                }

        }
}
```

```
package Vistas.dc.animacion3D
{
	// ************************************************************
	//
	//		Clase:	NodoDC
	//
	//		Patron de Programacion:
	//
	//			Auxiliar del modulo "MODELO" para simulacion en DC
	//
	//                                                                                septiembre
/ 2010
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;

	public class NodoDC extends MovieClip {

		// --- Contenedor ---
		public var contenedor:MovieClip;
		// --- Nodo Grafico ---
		public var nodoGrafico:Sprite;
		// --- Animacion grafica del Nodo ---
		public var animacion:Sprite;

		// --- Indice ---
		public var i:Number;

		// --- Coordenadas 3D ---
		public var x_n:Number;
		public var y_n:Number;
		public var z_n:Number;

		// --- Voltaje nodal ---
		public var voltajeNodal:Number;

		public function NodoDC(obj:Object):void {

			// --- Recepcion de parametros ---
			this.i = obj.i;
			this.x_n = obj.x_n;
			this.y_n = obj.y_n;
			this.z_n = obj.z_n;
			this.voltajeNodal = obj.voltajeNodal;

			// --- Asigancion de nombre del nodo ---
			this.name = "n"+i;

			// --- Asignacion de coordenadas en 3D ---
			this.x = x_n;
			this.y = y_n;
			this.z = z_n;
			trace(" ");
```

```
		// --- Construccion del nodo visual ---
		construccionNodoDCVisual();
	}

	public function construccionNodoDCVisual():void {

		// ------------------------------------------------------------
		//                              Construccion del nodo visual
		// ------------------------------------------------------------

		// Capa 0 : "contenedor"
		contenedor = new MovieClip();
		addChild(contenedor);

		// Capa 1 : "nodo grafico"
		nodoGrafico = new Sprite();
		contenedor.addChild(nodoGrafico);
		construyeNodo();

		// Capa 2 : "animar seleccion"
		animacion = new Sprite();
		construyeAnimacion();
	}

	public function construyeNodo():void {
		nodoGrafico.graphics.lineStyle(1,0x00ccff,1);
		nodoGrafico.graphics.beginFill(0x00ccff,0.5);
		nodoGrafico.graphics.drawCircle(0,0,3);
		nodoGrafico.graphics.endFill();
	}

	// ------------------------------------------------------------
	//
	//                              Animacion del Nodo Seleccionado
	//       ( "circulos" en forma de ondas azules emitidas )
	//
	// ------------------------------------------------------------

	public function construyeAnimacion():void {
		animacion.graphics.lineStyle(1,0x00ccff,1);
		animacion.graphics.drawCircle(0,0,3);
	}

	public function iniciarAnimacion():void {
		addChild(animacion);
		addEventListener(Event.ENTER_FRAME, onEnterFrame);
	}

	public function detenerAnimacion():void {
		removeEventListener(Event.ENTER_FRAME, onEnterFrame);
		removeChild(animacion);
	}

	public function onEnterFrame(event:Event):void {
		if (animacion.scaleX < 4) {
```

```
                                animacion.scaleX += 0.2;
                                animacion.scaleY += 0.2;
                        } else {
                                animacion.scaleX = 1;
                                animacion.scaleY = 1;
                        }
                }
        }
}

package Vistas.ac.animacion3D
{
        // ************************************************************
        //
        //      Clase:  NodoAC
        //
        //      Patron de Programacion:
        //
        //              Auxiliar del modulo "MODELO" para simulacion en AC
        //
        //                                                                      septiembre
/ 2010
        // ************************************************************

        // Flash principal
        import flash.display.*;
        // Eventos
        import flash.events.*;

        public class NodoAC extends MovieClip {

                // --- Contenedor ---
                public var contenedor:MovieClip;
                // --- Nodo Grafico ---
                public var nodoGrafico:Sprite;
                // --- Animacion grafica del Nodo ---
                public var animacion:Sprite;
                // --- Potenciales del nodo ---
                public var potenciales:Array;

                // --- Indice ---
                public var i:Number;

                // --- Coordenadas 3D ---
                public var x_n:Number;
                public var y_n:Number;
                public var z_n:Array;

                // --- Voltaje nodal complejo ---
                public var magnitud:Number;
                public var fase:Number;
                public var voltajeNodalPico:Number;

                public function NodoAC(obj:Object):void {

                        // --- Recepcion de parametros ---
```

```
            this.i = obj.i;
            this.x_n = obj.x_n;
            this.y_n = obj.y_n;
            this.z_n = obj.z_n;
            this.magnitud = obj.magnitud;
            this.fase = obj.fase;
            // ------------------------------------------------------------
            // Asignacion de una tabla de valores ("potenciales") de una
            // funcion cosenoidal defasada de acuerdo a la magnitud y fase
            //                              reportados por el simulador:
            //
            //                             potenciales = magnitud*coseno(2*pi*f + fase)
            //
            // ------------------------------------------------------------
            potenciales = z_n;
            this.voltajeNodalPico = obj.voltajeNodalPico;

            // --- Asigancion de nombre del nodo ---
            this.name = "n"+i;

            // --- Asignacion de coordenadas en 3D ---
            this.x = x_n;
            this.y = y_n;
            this.z = 0;

            // --- Construccion del nodo visual ---
            construccionNodoACVisual();
    }

    public function construccionNodoACVisual():void {

            // Capa 0 : "contenedor"
            contenedor = new MovieClip();
            addChild(contenedor);

            // Capa 1 : "nodo grafico"
            nodoGrafico = new Sprite();
            contenedor.addChild(nodoGrafico);
            construyeNodo();

            // Capa 2 : "animar seleccion"
            animacion = new Sprite();
            construyeAnimacion();
    }

    public function construyeNodo():void {
            nodoGrafico.graphics.lineStyle(1,0x00ccff,1);
            nodoGrafico.graphics.beginFill(0x00ccff,0.5);
            nodoGrafico.graphics.drawCircle(0,0,3);
            nodoGrafico.graphics.endFill();
    }

    // ------------------------------------------------------------
    //
    //  Asignacion de "potenciales" de forma dinamica mediante la
    //  funcion "onEnterFrame" generada en el modulo "motor3D_AC"
```

```
		//                                      ( oscilaciones en el espacio 3D )
		//
		// ---------------------------------------------------------

		public function asignaPotencial(t:Number):void {
			this.z = potenciales[t];
		}

		// ---------------------------------------------------------
		//
		//                              Animacion del Nodo Seleccionado
		//      ( "circulos" en forma de ondas azules emitidas )
		//
		// ---------------------------------------------------------

		public function construyeAnimacion():void {
			animacion.graphics.lineStyle(1,0x00ccff,1);
			animacion.graphics.drawCircle(0,0,3);
		}

		public function iniciarAnimacion():void {
			addChild(animacion);
			addEventListener(Event.ENTER_FRAME, onEnterFrame);
		}

		public function detenerAnimacion():void {
			removeEventListener(Event.ENTER_FRAME, onEnterFrame);
			removeChild(animacion);
		}

		public function onEnterFrame(event:Event):void {
			if (animacion.scaleX < 4) {
				animacion.scaleX += 0.2;
				animacion.scaleY += 0.2;
			} else {
				animacion.scaleX = 1;
				animacion.scaleY = 1;
			}
		}
	}
}
```

```
package com.red.simulador
{
	// ************************************************************
	//
	//	Clase:   simuladorDC
	//
	//	Patron de Programacion:
	//
	//	 Auxiliar de la clase "ModeloCircuito" para simulacion en DC
	//
	//                                                                                septiembre
/ 2010
	// ************************************************************
	//
	//	ENTRADA:
	//	- "redNodal" --> Descripcion del circuito en el formato:
	//                                                C[n] id: xx c: n1,n2 valor: xx tipo: DC o AC
	//                                          con:
	//                                                C[n]: --> Arreglo de descriptores de
componentes
	//                                                id: --> Identificador del componente
	//                                                c:  --> Arreglo de conexion de terminales
	//                                                n1,n2,n3: --> nodos de conexion del
componente
	//                                                valor: --> Valor del componente
	//                                                tipo: --> Tipo de fuente: "DC" o "AC"
	//
	//	SALIDA:
	//	- "voltajesNodales" --> Array: Valores todos los VOLTAJES NODALES
	//                                                                      del circuito
	//	- "corrientesRama" --> Array: Valores de las CORRIENTES DE RAMA
	//                                                                    obtenidos en la
simulacion en DC
	//
	// ************************************************************

	// Analisis_Nodal_Modificado_DC y Metodo_Gauss_Jordan_Real
	import com.red.simulador.dc.Analisis_Nodal_Modificado_DC;
	import com.red.simulador.dc.Metodo_Gauss_Jordan_Real;

	public class simuladorDC {

		// --- Descripcion del circuito ---
		public var C:Array;

		// --- Analisis_Nodal_Modificado_DC ---
		public var A:Array = new Array();
		public var xv:Array = new Array();
		public var b:Array = new Array();
		public var orden_Circuito:int;
		public var sistema:Analisis_Nodal_Modificado_DC;

		// --- Metodo_Gauss_Jordan_Real ---
		public var xvsol:Array;
		public var bsol:Array;
		public var resultado:Array;
```

```
		public var solucion:Metodo_Gauss_Jordan_Real;

		// --- Voltajes nodales ---
		public var voltajesNodales:Array;

		public function simuladorDC(redNodal:Array):void {

			// --- Recepcion del circuito ---
			this.C = redNodal;
			// --- Planteamiento del sistema de ecuaciones ---
			generaSistemaEcuaciones();
			// ---Solucion del sistema de ecuaciones ---
			resuelveSistemaEcuaciones();
		}

		// ------------------------------------------------------------
		//
		//                          Seccion de planteamiento y solucion del
		//                            sistema de ecuaciones del circuito
		//
		// ------------------------------------------------------------

		public function generaSistemaEcuaciones():void {

			// ------------------------------------------------------------
			//          Genera el sistema de ecuaciones "extendido" del circuito
			// ------------------------------------------------------------
			sistema = new Analisis_Nodal_Modificado_DC(C);
			A = sistema.A;
			xv = sistema.xv;
			b = sistema.b;

			// ------------------------------------------------------------
			//            Para fines de pruebas de funcionamiento del simulador
			// ------------------------------------------------------------
			//sistema.imprime_Matriz_y_Vectores();
		}

		public function resuelveSistemaEcuaciones():void {

			// ------------------------------------------------------------
			//                          Solucion del SISTEMA DE ECUACIONES por el
			//                          METODO DE GAUSS-JORDAN con numeros
reales
			// ------------------------------------------------------------
			var resultado:Array;
			var solucion:Metodo_Gauss_Jordan_Real = new
Metodo_Gauss_Jordan_Real(A,xv,b);
			xvsol = solucion.xv;
			bsol = solucion.b;

			// ------------------------------------------------------------
			//            Para fines de pruebas de funcionamiento del simulador
			// ------------------------------------------------------------
			//solucion.imprimeSolucion();
		}
```

```
		// ------------------------------------------------------------
		//
		//                          Seccion de recuperacion de voltajes nodales
		//                                          y corrientes de rama
		//
		// ------------------------------------------------------------

		public function obtenerVoltajesNodales():Array {

			// --- Obtencion de los volatjes nodales resultantes ---
			voltajesNodales = new Array();
			voltajesNodales.push(0);
			for (var i=0; i<bsol.length; i++) {
				if (xvsol[i].substr(0,1) == "V") {
					voltajesNodales.push(bsol[i]);
				}
			}

			return voltajesNodales;
		}

		public function obtenerCorrientesRama():Array {

			// --- Obtencion de las corrientes de rama resultantes ---
			var corrientesRama:Array = new Array();
			for (var i=0; i<bsol.length; i++) {
				if (xvsol[i].substr(0,1) != "V") {
					corrientesRama.push([xvsol[i].substring(2),bsol[i]]);
				}
			}

			return corrientesRama;
		}
	}
}
```

```
package com.red.simulador.dc
{
	// ************************************************************
	//
	//	Clase:   Analisis_Nodal_Modificado_DC
	//
	//	Patron de Programacion:
	//
	//				Auxiliar de la clase "simuladorDC"
	//
	//                                                                  febrero /
2010
	// ************************************************************
	//
	//	ENTRADA:
	//	- "C[n]" --> Arreglo de descriptores de componentes con el formato:
	//
	//                                  C[n] = id: xx c: n1,n2 valor: xx tipo: DC o AC
	//                                   con:
	//                                        id: --> Identificador del componente
	//                                        c:  --> Arreglo de conexion de terminales
	//                                        n1,n2,n3: --> nodos de conexion del
componente
	//                                        valor: --> Valor del componente
	//                                        tipo: --> Tipo de fuente: "DC" o "AC"
	//
	//	SALIDA:		Matriz aumentada "A" y vectores "xv" y "b"
	//
	//                                                  |A||xv| = |b|
	//                                  con:
	//                                        |A|  --> Number: Matriz aumentada
	//                                        |xv| --> String: Vector de variables a
calcular
	//                                                   (voltajes nodales y
corrientes de malla)
	//                                        |b|  --> Number: Vector de excitacion
	//
	//                                - Servicios de Impresion
	//                                      * imprime_Matriz_y_Vectores()
	//
	// ************************************************************

	 public class Analisis_Nodal_Modificado_DC {

		// --- Lista de componentes ---
		private var C:Array;
		// --- Lista de Nodos ---
		private var N:Array;

		// --- |A||xv| = |b| ---
		public var A:Array;
		public var xv:Array;
		public var b:Array;

		// --- Indices ---
		private var i:int;
```

```
private var j:int;
private var k:int;

// --- Auxiliares ---
private var numero_Ecuaciones_Adicionales:int;

public function Analisis_Nodal_Modificado_DC(_C:Array):void {

	// --- Recepcion del arreglo descriptor del circuito ---
	C = _C;

	// --- |A||xv| = |b| ---
	A = new Array();
	xv = new Array();
	b = new Array();

	construirMatrizAumentada();
}

private function construirMatrizAumentada():void {
	// --- Generar lista de nodos del circuito ---
	generarListaNodos();
	// --- Crear la matriz |A| y los vectores |xv| y |b| ---
	crear_Matriz_y_Vectores();
	// --- Inicializar la matriz |A| y los vectores |xv| y |b| ---
	inicializar_Matriz_y_Vectores();
}

// ------------------------------------------------------------
//
//                    Seccion de generacion de la red del circuito
//
// ------------------------------------------------------------

private function generarListaNodos():void {

	N = new Array();

	// --- Lectura de Nodos de la Lista de Componentes ---
	for (i=0; i<C.length; i++) {
		for (j=0; j<C[i].c.length; j++) {
			N.push(C[i].c[j]);
		}
	}

	// --- Ordenamiento ascendente del arreglo de nodos ---
	N.sort(Array.NUMERIC);

	// --- Eliminacion de nodos repetidos ---
	for (var j=0; j<N.length; j++) {
		for (var i=j+1; i<N.length; i++) {
			if (N[j] == N[i]) {
				N.splice(i,1);
				i -= 1;
			}
		}
```

```
		}
}

// ----------------------------------------------------------
//
// Seccion de creacion e inicializacion de la matriz y vectores:
//                                                    |A||xv| = |b|
//
// ----------------------------------------------------------

private function crear_Matriz_y_Vectores():void {

		// --- Auxiliares ---
		var renglones:Array;

		// --- Cuenta numero de fuentes ---
		numero_Ecuaciones_Adicionales = 0;
		for (i=0; i<C.length; i++) {
			switch(C[i].id.substr(0,1)) {

				// ------------------------------------
				//                    ECUACIONES ADICIONALES
				// ------------------------------------

				// Incremento por Fuente de VOLTAJE independiente
				case "V":
					numero_Ecuaciones_Adicionales++;
				break;
				// Incremento por Fuente de CORRIENTE
				// controlada por CORRIENTE
				case "F":
					numero_Ecuaciones_Adicionales++;
				break;
				// Incremento por CORTO CIRCUITO
				case "W":
					numero_Ecuaciones_Adicionales++;
				break;
				// Incremento por Fuente de CORRIENTE controlada
				// por VOLTAJE (TRANSISTOR)
				case "Q":
					numero_Ecuaciones_Adicionales++;
				break;
				// Incremento por AMPLIFICADOR OPERACIONAL
				case "A":
					numero_Ecuaciones_Adicionales++;
				break;
			}
		}

		// --- Inicializacion de la MATRIZ "A" y los vectores "b" y "xv" ---
		for (i=0; i<N.length+numero_Ecuaciones_Adicionales; i++) {
			renglones = new Array();
			for (j=0; j<N.length+numero_Ecuaciones_Adicionales; j++) {
				renglones.push(0);
			}
			// --- Creacion de la MATRIZ EXTENDIDA con valor cero ---
```

```
                    A.push(renglones);
                    // --- Creacion del VECTOR DE VARIABLES con valor cero ---
                    xv.push(0);
                    // --- Creacion del VECTOR DE EXCITACIONES con valor cero ---
                    b.push(0);
                }
        }

        private function inicializar_Matriz_y_Vectores():void {

                // --- Auxiliares ---
                var id:String;
                // --- Nodos de conexion ---
                var n1:int;
                var n2:int;
                // --- Nodos controladores ---
                var nc1:int;
                var nc2:int;
                // --- Valor de componente ---
                var valor:Number;
                // --- Contadores ---
                numero_Ecuaciones_Adicionales = 0;

                // --- Inicializacion de la Matriz "A y los vectores "b" y "xv" ---
                for (i=0; i<C.length; i++) {
                        id = C[i].id;
                        n1 = C[i].c[0];
                        n2 = C[i].c[1];
                        valor = C[i].valor;
                        if (C[i].tipo != undefined) {
                                tipo = C[i].tipo;
                        }

                        switch(id.substr(0,1)) {
                                // --- Resistencia ------------------------------------
                                case "R":
                                        A[n1][n1] += 1/valor;
                                        A[n2][n2] += 1/valor;
                                        A[n2][n1] -= 1/valor;
                                        A[n1][n2] -= 1/valor;
                                break;
                                // --- Capacitor -------------------------------------
                                case "C":
                                        A[n1][n1] += 0;
                                        A[n2][n2] += 0;
                                        A[n2][n1] -= 0;
                                        A[n1][n2] -= 0;
                                break;
                                // --- Fuente de Voltaje independiente de DC ----------
                                case "V":
                                        numero_Ecuaciones_Adicionales++;
                                        A[n1][A[0].length-numero_Ecuaciones_Adicionales]
= -1;
                                        A[n2][A[0].length-numero_Ecuaciones_Adicionales]
= 1;
```

```
                                        A[A[0].length-numero_Ecuaciones_Adicionales][n1]
= -1;
                                        A[A[0].length-numero_Ecuaciones_Adicionales][n2]
= 1;
                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =
"I_"+id;
                                        if (tipo == "AC") {
                                                // Analisis en AC
                                                // --- Pasivacion de fuentes de voltaje --------
-
                                                b[A[0].length-
numero_Ecuaciones_Adicionales] = 0;
                                        } else {
                                                // Analisis en DC
                                                b[A[0].length-
numero_Ecuaciones_Adicionales] = valor;
                                        }
                                break;
                                // --- Fuente de Corriente independiente de DC ----------
                                case "I":
                                        b[n1] -= valor;
                                        b[n2] -= -valor;
                                break;
                                // --- Fuente de CORRIENTE controlada por VOLTAJE ------
-
                                case "G":
                                        nc1 = C[i].c[2];
                                        nc2 = C[i].c[3];
                                        A[n1][nc1] += valor;
                                        A[n1][nc2] += -valor;
                                        A[n2][nc1] += -valor;
                                        A[n2][nc2] += valor;
                                break;
                                // --- Fuente de CORRIENTE controlada por CORRIENTE -
----
                                case "F":
                                        nc1 = C[i].c[2];
                                        nc2 = C[i].c[3];
                                        numero_Ecuaciones_Adicionales++;
                                        A[nc1][A[0].length-
numero_Ecuaciones_Adicionales] = 1;
                                        A[nc2][A[0].length-
numero_Ecuaciones_Adicionales] = -1;
                                        A[n1][A[0].length-numero_Ecuaciones_Adicionales]
= valor;
                                        A[n2][A[0].length-numero_Ecuaciones_Adicionales]
= -valor;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][nc1] = 1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][nc2] = -1;
                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =
"I_"+id;
                                break;
                                // --- CORTO CIRCUITO ----------------------------------
                                case "W":
```

```
                                                numero_Ecuaciones_Adicionales++;
                                                A[n1][A[0].length-numero_Ecuaciones_Adicionales]
= 1;
                                                A[n2][A[0].length-numero_Ecuaciones_Adicionales]
= -1;
                                                A[A[0].length-numero_Ecuaciones_Adicionales][n1]
= 1;
                                                A[A[0].length-numero_Ecuaciones_Adicionales][n2]
= -1;
                                                A[A[0].length-
numero_Ecuaciones_Adicionales][A[0].length-numero_Ecuaciones_Adicionales] = valor;
                                                xv[A[0].length-numero_Ecuaciones_Adicionales] =
"I_"+id;
                                        break;
                                        // --- AMPLIFICADOR OERACIONAL "A" --- FVCV ----------
--
                                        case "A":       // n1(V+); n2(V-); nc1(Vo)
                                                nc1 = C[i].c[2];
                                                numero_Ecuaciones_Adicionales++;
                                                A[nc1][A[0].length-
numero_Ecuaciones_Adicionales] = -1;
                                                A[A[0].length-numero_Ecuaciones_Adicionales][n1]
= 1;
                                                A[A[0].length-numero_Ecuaciones_Adicionales][n2]
= -1;
                                                xv[A[0].length-numero_Ecuaciones_Adicionales] =
"I_"+id;
                                        break;
                                }
                        }

                        // Cortar renglon "0" de "A", "xv" y "b", correspondiente
                        // al nodo de "tierra"
                        A.splice(0,1);
                        xv.splice(0,1);
                        b.splice(0,1);
                        // --- Actualiza el vector "xv" ---
                        for (i=0; i<N.length-1; i++) {
                                xv[i] = "V"+(i+1);
                        }
                        // Cortar columna "0" de "A" , correspondiente
                        // al nodo de "tierra"
                        for (i=0; i<A.length; i++) {
                                A[i].splice(0,1);
                        }

                }

                // ------------------------------------------------------
                //
                //                      Seccion de impresion de resultados:
                //                                      |A||xv| = |b|
                //
                // ------------------------------------------------------

                public function imprime_Matriz_y_Vectores():void {
```

```
                        // ------------------------------------------------------------
                        //          |A||xv| = |b|
                        //                  con:
                        //                  |A| --> Matriz aumentada
                        //                  |xv| --> Vector de variables a calcular
                        //                                  (voltajes nodales y corrientes de malla)
                        //                  |b| --> Vector de excitacion
                        // ------------------------------------------------------------
                        trace(" ");
                        trace("*** MATRIZ [A] ***");
                        for (i=0; i<A.length; i++) {
                                trace(" "+A[i]);
                        }

                        trace(" ");
                        trace("*** VECTOR [xv] ***");
                        for (i=0; i<xv.length; i++) {
                                trace(" "+xv[i]);
                        }

                        trace(" ");
                        trace("*** VECTOR [b] ***");
                        for (i=0; i<b.length; i++) {
                                trace(" "+b[i]);
                        }
                }

        }
}

package com.red.simulador.dc
{
        // ************************************************************
        //
        //          Clase:  Metodo_Gauss_Jordan_Real
        //
        //          Patron de Programacion:
        //
        //                                  Auxiliar de la clase "simuladorDC"
        //
        //                                                                                  febrero /
2010
        // ************************************************************
        //
        //          ENTRADA:        Matriz aumentada "A" y vectores "xv" y "b"
        //
        //                                                                  |A||xv| = |b|
        //                                          con:
        //                                                  |A|  --> Number: Matriz aumentada
        //                                                  |xv| --> String: Vector de variables a
calcular
        //                                                                  (voltajes nodales y
corrientes de malla)
        //                                                  |b|  --> Number:Vector de excitacion
        //
```

```
	//		SALIDA:			|xv| = |b = solucion|
	//							con:
	//								|xv| --> String: Vector de variables a
calcular
	//									(voltajes nodales y
corrientes de malla)
	//								|b|  --> Number: Vector solucion
	//
	//					- Servicios de Impresion
	//						* imprimeSolucion()
	//
	// ************************************************************

	 public class Metodo_Gauss_Jordan_Real {

		// --- |A||xv| = |b| ---
		private var A:Array;
		public var xv:Array;
		public var b:Array;

		// --- Indices ---
		private var i:int;
		private var j:int;
		private var k:int;

		// --- Auxiliares ---
		private var permutaciones:int;

		public function Metodo_Gauss_Jordan_Real(_A:Array,_xv:Array,_b:Array):void {

			// --- Recepcion de la matriz y los vectores ---
			A = _A;
			xv = _xv;
			b = _b;

			resolverSistemaEcuaciones();
		}

		private function resolverSistemaEcuaciones():void {
			solucionGaussJordan();
		}

		private function solucionGaussJordan():void {

			// --- Auxiliares ---
			var pivote:Number;
			var elemento:Number;
			var renglon:int;
			var r:Number;
			var s:Number;
			var temp:Number;

			permutaciones = 0;

			// --- Creacion de la MATRIZ TRIANGULAR SUPERIOR ---
			// --- Proceso DIRECTO ---
```

```
for (i=0; i<A.length; i++) {
	pivote = Math.abs(A[i][i]);
	renglon = i;

	// --- Busqueda del mayor elemento de la matriz ---
	for (j=i+1; j<A.length; j++) {
		elemento = Math.abs(A[j][i]);
		if (pivote < elemento) {
			pivote = elemento;
			renglon = j;
		}
	}

	// --- Decision para intercambio de renglones --
	if (renglon != i) {
		for (j=i; j<A.length; j++) {
			// --- Intercambio en la matriz "A" ---
			temp = A[renglon][j];
			A[renglon][j] = A[i][j];
			A[i][j] = temp;
		}
		// --- Intercambio en el vector "b" ---
		temp = b[renglon];
		b[renglon] = b[i];
		b[i] = temp;
		// --- Registra numero de permutaciones ---
		permutaciones++;
	}

	// --- Actualizacion del pivote ---
	pivote = A[i][i];

	// --- Deteccion de matriz singular ---
	if (pivote == 0) {
		trace("*********************************");
		trace("*   ERROR: Matriz singular    *");
		trace("*              <PROCESO DIRECTO>                    *");
	   trace("         pivote = 0          *");
		trace("*********************************");
		break;
	} else {
		// --- Calculo de los elementos de la nueva matriz "A" ---
		for (j=i+1; j<A.length; j++) {
			r = A[j][i]/pivote;
			for (k=i; k<A[i].length; k++) {
				A[j][k] -= r*A[i][k];
			}
			b[j] -= r*b[i];
		}
	}
}


// --- Proceso INVERSO ---
for (i=A.length-1; i>0; i--) {
	pivote = A[i][i];
```

```
                        if (pivote == 0) {
                                trace("*********************************");
                                trace("*   ERROR: Matriz singular    *");
                                trace("*              <PROCESO INVERSO>                    *");
                            trace("         pivote = 0          *");
                                trace("*********************************");
                                break;
                        } else {
                                for (j=i-1; j>=0; j--) {
                                        r = A[j][i]/pivote;
                                        for (k=i; k>0; k--) {
                                                A[j][k] -= r*A[i][k];
                                        }
                                        b[j] -= r*b[i];
                                }
                        }
                }

                // --- Solucion FINAL ---
                for (i=0; i<b.length; i++) {
                        b[i] /= -A[i][i];
                }
        }

        public function imprimeSolucion():void {

                // ------------------------------------------------------------
                //      |xv| = |b = solucion|
                //              con:
                //              |xv| --> Vector de variables a calcular
                //                              (voltajes nodales y corrientes de malla)
                //              |b| --> Vector solucion
                // ------------------------------------------------------------
                trace(" ");
                trace("*** RESULTADOS ***");
                for (i=0; i<b.length; i++) {
                        trace(" "+xv[i]+" = "+b[i]);
                }
        }

    }
}
```

```
package com.red.simulador
{
	// ************************************************************
	//
	//	Clase:   simuladorAC
	//
	//	Patron de Programacion:
	//
	//	 Auxiliar de la clase "ModeloCircuito" para simulacion en AC
	//
	//                                                                                septiembre
/ 2010
	// ************************************************************
	//
	//	ENTRADA:
	//	- "redNodal" --> Descripcion del circuito en el formato:
	//                                                C[n] id: xx c: n1,n2 valor: xx tipo: DC o AC
	//                                          con:
	//                                                C[n]: --> Arreglo de descriptores de
componentes
	//                                                id: --> Identificador del componente
	//                                                c:  --> Arreglo de conexion de terminales
	//                                                n1,n2,n3: --> nodos de conexion del
componente
	//                                                valor: --> Valor del componente
	//                                                tipo: --> Tipo de fuente: "DC" o "AC"
	//
	//	SALIDA:
	//  - "LD" --> Object: Creacion y llenado del objeto:
	//                                             "Generador_Lista_Datos"
	//	* "LF" --> Array: Valores de Frecuencia en variacion logaritmica
	//                                          (de "fmin" a "fmax")
	//	* "resultado" --> Object: Valores de voltajes y corrientes complejos
	//                                             obtenidos en la simulacion en AC con
	//                                                      formato rectangular (a una
frecuencia
	//                                                      especifica). Cada "resultado" se
procesa para
	//                                                      generar diferentes formatos, y se
guarda en
	//                                                      el objeto derivado de la clase
	//                                                      "Generador_Lista_Datos"
	//
	//	- "voltajesNodalesComplejos" --> Array: Valores de MAGNITUD y FASE de los
	//
"voltajesNodalesComplejos" a la
	//                                                                        frecuencia central
del barrido logaritmico,
	//                                                                        de todas las
variables calculadas por el
	//                                                                  simulador en AC
	//
	// ************************************************************

	import flash.geom.Point;
```

```
// Analisis_Nodal_Modificado_Complejo y Metodo_Gauss_Jordan_Complejo
import com.red.simulador.ac.Analisis_Nodal_Modificado_Complejo;
import com.red.simulador.ac.Metodo_Gauss_Jordan_Complejo;

// Fuente de datos
import Modelo.Generador_Lista_Datos;

// Matematicas
import com.red.simulador.ac.Operaciones_Numeros_Complejos;

public class simuladorAC {

	// --- Descripcion del circuito ---
	public var C:Array;

	// --- Analisis_Nodal_Modificado_Complejo ---
	public var A:Array = new Array();
	public var xv:Array = new Array();
	public var b:Array = new Array();
	public var orden_Circuito:int;
	public var sistema:Analisis_Nodal_Modificado_Complejo;

	// --- Metodo_Gauss_Jordan_Complejo ---
	public var xvsol:Array;
	public var bsol:Array;
	public var resultado:Array;
	public var solucion:Metodo_Gauss_Jordan_Complejo;

	// --- Generador_Lista_Datos ---
	public var LD:Generador_Lista_Datos;
	public var barridoFrecuencia:Array;
	public var voltajesNodalesComplejos:Array;
	public var indiceFrecuenciaCentral:Number;

	public function simuladorAC(redNodal:Array):void {

		this.C = redNodal;

		// --- Operaciones complejas auxiliares ---
		operacion_compleja = new Operaciones_Numeros_Complejos();

		// Genera valores de barrido de frecuencia (Hz)
		//                              entre "fmin" y "fmax"
		barridoFrecuencia = new Array();

		// Genera Lista de Datos con barrido de frecuencia
		LD = new Generador_Lista_Datos();
		var fmin:Number = 0.001;        // Hz
		var fmax:Number = 1000;         // Hz
		var numMuestras:Number = 400;
		barridoFrecuencia = LD.generarBarridoFrecuencia(fmin,fmax,

		numMuestras);

		indiceFrecuenciaCentral = Math.round(numMuestras/2);
	}
```

```
		public function simularAC_Frecuencia_Especifica(redNodal:Array,

		f:Number):void {

			// ------------------------------------------------------------
			//                              Analisis_Nodal_Modificado_Complejo
			// ------------------------------------------------------------
			sistema = new Analisis_Nodal_Modificado_Complejo(redNodal);
			sistema.analizar("AC");

			// ------------------------------------------------------------
			//          Para fines de pruebas de funcionamiento del simulador
			// ------------------------------------------------------------
			//sistema.imprime_Matriz_y_Vectores_FormatoMatricial();
			//sistema.imprime_Matriz_y_Vectores();

			// ------------------------------------------------------------
			//                              Solucion del SISTEMA DE ECUACIONES por
			//                               el METODO DE GAUSS-JORDAN (complejo)
			// ------------------------------------------------------------
			solucion = new Metodo_Gauss_Jordan_Complejo(sistema);
			solucion.calcula_Voltajes_Corrientes_AC(f);

			// ------------------------------------------------------------
			//          Para fines de pruebas de funcionamiento del simulador
			// ------------------------------------------------------------
			//solucion.imprime_Solucion_Matriz_Ab();
		}

		public function simular_Barrido_Frecuencia(redNodal:Array):void {

			// ------------------------------------------------------------
			//                              Analisis_Nodal_Modificado_Complejo
			// ------------------------------------------------------------
			sistema = new Analisis_Nodal_Modificado_Complejo(redNodal);
			sistema.analizar("AC");

			// ------------------------------------------------------------
			//                              Solucion del SISTEMA DE ECUACIONES por
			//                               el METODO DE GAUSS-JORDAN (complejo)
			// ------------------------------------------------------------
			solucion = new Metodo_Gauss_Jordan_Complejo(sistema);
			// Calculo de voltajes y corrientes complejos (en formato rectangular)
			// para cada valor de frecuencia con barrido logaritmico
			var resultado:Object;
			for (indiceFrecuencia in barridoFrecuencia) {
				// ------------------------------------------------------------
				// Barrido de frecuencias "barridoFrecuencia[indiceFrecuencia]"
				// ------------------------------------------------------------
				resultado = solucion.calcula_Voltajes_Corrientes_AC(

barridoFrecuencia[indiceFrecuencia]);
				LD.generarListaSimulaciones(indiceFrecuencia,resultado);
			}
```

```
                // ------------------------------------------------------------
                //          Para fines de pruebas de funcionamiento del simulador
                // ------------------------------------------------------------
                /*trace(" ");
                trace("simuladorAC - Metodo_Gauss_Jordan_Complejo::
                        Solucion a frecuencia especifica");
                solucion.imprime_Solucion_Matriz_Ab();*/

        }

        public function validarSimulacionAC():Boolean {

                // ------------------------------------------------------------
                //                  Verifica que no existan errores en la simulacion
                //          Prueba que los resultados obtenidos sean "numericos" y
                //                                                      "finitos"
                // ------------------------------------------------------------
                var simulacionValida = true;
                for (numeroDato=0; numeroDato<LD.LD.length; numeroDato++) {
                        for(indiceFrecuencia=0; indiceFrecuencia<LD.LF.length;
                                indiceFrecuencia++) { // Frecuencia
                                if (isNaN(LD.LD[numeroDato][1][indiceFrecuencia].x) ||

        !isFinite(LD.LD[numeroDato][1][indiceFrecuencia].x) ||
                                                isNaN(LD.LD[numeroDato][1][indiceFrecuencia].y)
||

        !isFinite(LD.LD[numeroDato][1][indiceFrecuencia].y) ||
                                                isNaN(LD.LD[numeroDato][2][indiceFrecuencia].x)
||

        !isFinite(LD.LD[numeroDato][2][indiceFrecuencia].x) ||
                                                isNaN(LD.LD[numeroDato][2][indiceFrecuencia].y)
||

        !isFinite(LD.LD[numeroDato][2][indiceFrecuencia].y) ||
                                                isNaN(LD.LD[numeroDato][3][indiceFrecuencia]) ||
                                                !isFinite(LD.LD[numeroDato][3][indiceFrecuencia])
||
                                                isNaN(LD.LD[numeroDato][4][indiceFrecuencia]) ||
                                                !isFinite(LD.LD[numeroDato][4][indiceFrecuencia]))
{
                                                simulacionValida = false;
                                        }
                                }
                        }
                        return simulacionValida;
                }

                public function obtenerVoltajesNodalesComplejos():Array {

                        // ------------------------------------------------------------
                        //                  Obtiene los valores de MAGNITUD y FASE de los
                        //          "voltajesNodalesComplejos" a la frecuencia central
                        //                   (logaritmica) del barrido de frecuencia "LF"
                        // ------------------------------------------------------------
```

```
                        voltajesNodalesComplejos = new Array();

                        for (i=0; i<LD.LD.length; i++) {
                                if (LD.LD[i][0][indiceFrecuenciaCentral].substr(0,1) == "V") {
                                        voltajesNodalesComplejos.push(LD.LD[i][2]
                                                                        [indiceFrecuenciaCentral]);
                                }
                        }
                        return voltajesNodalesComplejos;
                }

                public function obtenerCorrientesRama():Array {

                        var corrientesRama:Array = new Array();
                        for (var i=0; i<bsol.length; i++) {
                                if (xvsol[i].substr(0,1) != "V") {
                                        corrientesRama.push([xvsol[i].substring(2),bsol[i]]);
                                }
                        }
                        return corrientesRama;
                }

        }
}

package Modelo
{
        // ************************************************************
        //
        //      Clase:  Generador_Lista_Datos
        //
        //      Patron de Programacion:
        //
        //       Auxiliar de la clase "ModeloCircuito" para simulacion en AC
        //
        //                                                                              septiembre
/ 2010
        // ************************************************************
        //
        //      ENTRADA:      - "voltajesNodalesComplejos" a una frecuencia especifica
        //
        //      SALIDA:
        //      - LF --> Array: Valores de Frecuencia en variacion logaritmica
        //                                      (de "fmin" a "fmax")
        //      - LD --> Array: Valores de voltajes y corrientes obtenidos en la
        //                                      simulacion en AC con el siguiente formato:
        //      * LD[i][0][indiceFrecuencia] = dato.xv[i]:          String (identificador)
        //      * LD[i][1][indiceFrecuencia] = formaRectagular: Point(x, jy}
        //      * LD[i][2][indiceFrecuencia] = Fasor:                       Point(MAGNITUD, FASE}
(Polar)
        //      * LD[i][3][indiceFrecuencia] = Magnitud_dB:             Number
        //      * LD[i][4][indiceFrecuencia] = Fase_Grados:     Number
        //
        //      * Funciones Publicas
        //              -
generarBarridoFrecuencia(f_min=0.001,f_max=1000,numero_Muestras=400)
```

```
		//
		//		* Servicios de Impresion
		//			- imprimir_Barrido_Frecuencias() -> Imprime valores de frecuencia
		//                                                                                                  (de
"fmin" a "fmax")
		//			- imprimir_LD() --> Imprime todos los datos del Array LD
		//
		// ************************************************************

		// Matematicas
		import flash.geom.Point;
		import com.red.simulador.ac.Operaciones_Numeros_Complejos;

		 public class Generador_Lista_Datos {

			// --- Lista de Datos ---
			public var LD:Array;
			// --- Lista de Frecuencias ---
			public var LF:Array;
			// --- Frecuencias en Hz ---
			public var f_max:Number;		// Valor maximo de frecuencia
			public var f_min:Number;		// Valor minimo de frecuencia
			// --- Numero de muestras a generar ---
			public var numero_Muestras:Number;
			private var primeraEvaluacionFrecuencia:Boolean;

			// --- Indices ---
			private var i:int;
			private var k:int;
			private var m:int;
			private var n:int;

			// ---  Variables de conversion de barrido lineal a logaritmico ---
			public var numero_Decadas_Frecuencia:Number;
			private var convierte_ValoresFrecuencia_aDecadas:Number;
			private var convierte_Frecuencia_aLogaritmica:Number;

			// --- Constante de conversion de logNatural a logDecimal ---
			static const LN_LOG:Number = 0.434294;

			// --- Variables compleja "s" ---
			private var s:Point;
			// --- Valores de "f" en variacion logaritmica ---
			private var Afrecuencia:Array;

			// --- Variables fasoriales ---
			private var Fasor:Point;
			private var Magnitud_dB:Number;
			private var Fase_Grados:Number;

			// --- Matematicas: Operaciones con Numeros Complejos ---
			private var operacion_compleja:Operaciones_Numeros_Complejos;

			public function Generador_Lista_Datos():void {

				// --- Lista de Frecuencias ---
```

```
                LF = new Array();
                // --- Lista de Datos ---
                LD = new Array();

                // --- Operaciones con Numeros Complejos ---
                operacion_compleja = new Operaciones_Numeros_Complejos();

                // --- Flag ---
                primeraEvaluacionFrecuencia = true;

        }

        // ---------------------------------------------------------
        //
        //      Seccion de GENERACION DE VALORES DE FRECUENCIA CON
BARRIDO
        //                                                  LOGARITMICO
        //
        // ---------------------------------------------------------

        public function generarBarridoFrecuencia(f_min:Number=0.001,

f_max:Number=1000,

numero_Muestras:Number=400):Array {
                this.f_min = f_min;
                this.f_max = f_max;

                // --- Numero de puntos a calcular para el barrido en frecuencia ---
                this.numero_Muestras = numero_Muestras;

                // --- Frecuencia ---
                var frecuencia:Number;

                // Calculo de parametros de conversion de:
                // barrido lineal --> "indice" del arreglo "Afrecuencia[inidce]"
                // a barrido logarítmico --> valores del arreglo "Afrecuencia"
                inicializar_Parametros_CalculoBarridoFrecuencia();

                // ---------------------------------------------------------
                // Genera valores de frecuencia (Hz) con variacion logaritmica
                //                                              en el arreglo "LF"
                // ---------------------------------------------------------
                for (i=0; i<numero_Muestras; i++) {
                        // Generacion de barrido logaritmico para la frecuencia
                        // "frecuencia"
                        frecuencia = Math.pow(10,(i+convierte_Frecuencia_aLogaritmica)/

convierte_ValoresFrecuencia_aDecadas);
                        // Guardado de los valores obtenidos de "frecuencia"
                        // en el arreglo "Afrecuencia"
                        LF.push(frecuencia);
                }

                return LF;
        }
```

```
		private function inicializar_Parametros_CalculoBarridoFrecuencia():void {

			// ---------------------------------------------------------
			// Calculo de parametros para generar valores de frecuencia
			//		que se puedan graficar en un eje de frecuencia
			//			logaritmico (por decadas)
			// ---------------------------------------------------------
			numero_Decadas_Frecuencia =
Math.round(LN_LOG*Math.log(f_max/f_min));
			convierte_ValoresFrecuencia_aDecadas = numero_Muestras/

numero_Decadas_Frecuencia;
			convierte_Frecuencia_aLogaritmica =
convierte_ValoresFrecuencia_aDecadas*

	LN_LOG*Math.log(f_min);
		}

		// ---------------------------------------------------------
		//
		//	Seccion de GENERACION DE VALORES DE TODAS LAS VARIABLES
		//  REPORTADAS POR EL SIMULADOR EN AC (voltajes y corrientes),
		//   PARA CADA VALOR DE FRECUENCIA CON BARRIDO LOGARITMICO
		//
		// ---------------------------------------------------------

		public function generarListaSimulaciones(indiceFrecuencia:Number,

dato:Object):void {

			// ---------------------------------------------------------
			// Para cada valor de frecuencia del arreglo "LF", simula el
			// circuito en "AC", convierte los resultados del simulador
			// a diferentes formatos y guarda los resultados para cada
			//  "voltaje nodal" y cada "corriente de malla",
			// en el arreglo "LD"
			// ---------------------------------------------------------
			var xP:Number;
			var yP:Number;
			var formaRectagular:Point;

			if (primeraEvaluacionFrecuencia) {

				// --- Definicion de datos para el "nodo de tierra" ---
				LD[0] = new Array();
				LD[0][0] = new Array();		// Identificador
				LD[0][1] = new Array();		// Formato Rectangular
				LD[0][2] = new Array();		// Formato Polar (Fasor)
				LD[0][3] = new Array();		// Magntiud dB
				LD[0][4] = new Array();		// Fase Grados

				// Definicion de datos para cada variable contenida
				// en "Voltajes_Nodales_Extendido", las cuales representan
				// cada uno de los resultados generados por el simulador en AC
				for (i in dato.Voltajes_Nodales_Extendido) {
```

```
                                LD[i+1] = new Array();
                                LD[i+1][0] = new Array();          // Identificador
                                LD[i+1][1] = new Array();          // Formato Rectangular
                                LD[i+1][2] = new Array();          // Formato Polar (Fasor)
                                LD[i+1][3] = new Array();          // Magntiud dB
                                LD[i+1][4] = new Array();          // Fase Grados
                        }
                        primeraEvaluacionFrecuencia = false;
                }

                // Evaluacion del voltaje nodal de "tierra" para cada frecuencia ---
                xP = 0;
                yP = 0;
                formaRectagular = new Point(xP, yP);
                LD[0][0].push("V0");                               // Identificador
                LD[0][1].push(formaRectagular);                    // Formato Rectangular
                calularMagnitudFase(formaRectagular);
                LD[0][2].push(Fasor);                              // Formato Polar (Fasor)
                LD[0][3].push(Magnitud_dB);                        // Magntiud dB
                LD[0][4].push(Fase_Grados);                        // Fase Grados

                // Evaluacion de las variables contenidas en
                // "Voltajes_Nodales_Extendido" para cada frecuencia
                for (i in dato.Voltajes_Nodales_Extendido) {
                        xP = dato.Voltajes_Nodales_Extendido[i].x;
                        yP = dato.Voltajes_Nodales_Extendido[i].y;
                        formaRectagular = new Point(xP, yP);
                        LD[i+1][0].push(dato.xv[i]);                       // Identificador
                        LD[i+1][1].push(formaRectagular);                  // Formato
Rectangular
                        calularMagnitudFase(formaRectagular);
                        LD[i+1][2].push(Fasor);                                    // Formato
Polar (Fasor)
                        LD[i+1][3].push(Magnitud_dB);                      // Magntiud dB
                        LD[i+1][4].push(Fase_Grados);                      // Fase Grados
                }

        }

        private function calularMagnitudFase(formaRectagular:Point):void {

                // ------------------------------------------------------------
                //  Convertidor de "formato rectangular" a "formato fasorial"
                // ------------------------------------------------------------
                // --- Variables para operaciones complejas ---
                var forma_Polar:Point;

                forma_Polar = operacion_compleja.convierte_a_Polar(formaRectagular);
                operacion_compleja.ajusta_VALOR(forma_Polar);
                if (forma_Polar.x == 0) {
                        Magnitud_dB = 0;
                        Fase_Grados = 0;
                        forma_Polar.y = 0;
                        Fasor = forma_Polar;
                } else {
```

```
                                        Magnitud_dB =
Math.round(20*LN_LOG*Math.log(forma_Polar.x)*100)/100;
                                        Fase_Grados = Math.round(forma_Polar.y*100)/100;
                                        Fasor = forma_Polar;
                                }
                        }

                        // ----------------------------------------------------------
                        //
                        //          Seccion de IMPRESION
                        //    Para fines de pruebas de funcionamiento del simulador
                        //
                        // ----------------------------------------------------------

                        public function imprimir_Barrido_Frecuencias():void {
                                trace("  ");
                                trace("*****************************************************");
                                trace("              --- <LF> LISTA DE FRECUENCIAS ---");
                                trace("*****************************************************");
                                trace(" ");

                                for(i=0; i<LF.length; i++) {
                                        trace("       "+i+"\t"+LF[i]);
                                }
                        }

                        public function imprimir_LD():void {
                                trace("   ");
                                trace("*****************************************************");
                                trace("                          --- LISTA DE DATOS ---");
                                trace("     FRECUENCIA, RECTANGULAR, FASOR, MAGNITUD Y FASE
");
                                trace("*****************************************************");
                                trace(" ");

                                for(indiceFrecuencia=0; indiceFrecuencia<LF.length; indiceFrecuencia++) {

        trace("+++++++++++++++++++++++++++++++++++++++++++++++++");
                                        trace(indiceFrecuencia+"\tFrecuencia= "+LF[indiceFrecuencia]);
                                        for (numeroDato=0; numeroDato<LD.length; numeroDato++) {
                                                trace(LD[numeroDato][0][indiceFrecuencia]+"\t"+
                                                        LD[numeroDato][1][indiceFrecuencia]+"\t"+
                                                        LD[numeroDato][2][indiceFrecuencia]+"\t"+
                                                        LD[numeroDato][3][indiceFrecuencia]+"\t"+
                                                        LD[numeroDato][4][indiceFrecuencia]);
                                        }
                                }
                        }
                }
}
```

```
package com.red.simulador.ac
{
	// ************************************************************
	//
	//	Clase:   Analisis_Nodal_Modificado_Complejo
	//
	//	Patron de Programacion:
	//
	//				Auxiliar de la clase "simuladorAC"
	//
	//                                                                septiembre
/ 2010
	// ************************************************************
	//
	//	ENTRADA:
	//	- "C[n]" --> Arreglo de descriptores de componentes con el formato:
	//
	//                                  C[n] = id: xx c: n1,n2 valor: xx tipo: DC o AC
	//                                   con:
	//                                        id: --> Identificador del componente
	//                                        c:  --> Arreglo de conexion de terminales
	//                                        n1,n2,n3: --> nodos de conexion del
componente
	//                                        valor: --> Valor del componente
	//                                        tipo: --> Tipo de fuente: "DC" o "AC"
	//
	//	SALIDA:		Matriz compleja aumentada "A", vector "xv"
	//                              y vector complejo "b"
	//
	//                                                      |A||xv| = |b|
	//                                  con:
	//                                        |A|  --> Point: Matriz compleja aumentada
	//                                        |xv| --> String: Vector de variables a
calcular
	//                                                       (voltajes nodales y
corrientes de malla)
	//                                        |b|  --> Point: Vector complejo de excitacion
	//
	//                              - Servicios de Impresion
	//                                      * imprime_Matriz_y_Vectores()
	//                                      * imprime_Matriz_y_Vectores_FormatoMatricial()
	//
	// ************************************************************

	// --- Matematicas ---
	import flash.geom.Point;

	 public class Analisis_Nodal_Modificado_Complejo {

		// --- Lista de componentes ---
		public var C:Array;
		// --- Lista de Nodos ---
		public var N:Array;

		// --- |A||xv| = |b| ---
		public var A:Array;
```

```
public var xv:Array;
public var b:Array;

// --- Indices ---
private var i:int;
private var j:int;
private var k:int;

// --- Tipo de analisis ---
public var analisis:String;

// --- Auxiliares ---
private var numero_Ecuaciones_Adicionales:int;

public function Analisis_Nodal_Modificado_Complejo(_C:Array):void {

	// --- Recepcion del arreglo descriptor del circuito ---
	C = _C;
}

public function analizar(_analisis:String):void {

	// --- Tipo de analisis ---
	analisis = _analisis;

	// --- |A||xv| = |b| ---
	A = new Array();
	xv = new Array();
	b = new Array();

	// --- Generar lista de nodos del circuito ---
	generarListaNodos();
	// --- Crear la matriz |A| y los vectores |xv| y |b| ---
	crear_Matriz_y_Vectores();
	// --- Inicializar la matriz |A| y los vectores |xv| y |b| ---
	inicializar_Matriz_y_Vectores();
}

// ------------------------------------------------------------
//
//                    Seccion de generacion de la red del circuito
//
// ------------------------------------------------------------

private function generarListaNodos():void {

	N = new Array();

	// --- Lectura de Nodos de la Lista de Componentes ---
	for (i=0; i<C.length; i++) {
		N.push(C[i].c[0]);
		N.push(C[i].c[1]);
	}

	// --- Ordenamiento ascendente del arreglo de nodos ---
	N.sort(Array.NUMERIC);
```

```
		// --- Eliminacion de nodos repetidos ---
		for (var j=0; j<N.length; j++) {
			for (var i=j+1; i<N.length; i++) {
				if (N[j] == N[i]) {
					N.splice(i,1);
					i -= 1;
				}
			}
		}
	}
	// ----------------------------------------------------------
	//
	// Seccion de creacion e inicializacion de la matriz y vectores:
	//                                                    |A||xv| = |b|
	//
	// ----------------------------------------------------------

	private function crear_Matriz_y_Vectores():void {

		// --- Auxiliares ---
		var renglones:Array;
		var datos:Array;

		// --- Cuenta numero de fuentes de voltaje ---
		numero_Ecuaciones_Adicionales = 0;

		for (i=0; i<C.length; i++) {
			switch(C[i].id.substr(0,1)) {

				// -------------------------------------
				//                  ECUACIONES ADICIONALES
				// -------------------------------------

				// Incremento por calculo de CORRIENTE en el
CAPACITOR ---
				/*case "C":
					numero_Ecuaciones_Adicionales++;
				break;*/
				// Incremento por Fuente de VOLTAJE independiente ---
				case "V":
					numero_Ecuaciones_Adicionales++;
				break;
				// Incremento por Fuente de CORRIENTE controlada
				// por CORRIENTE
				case "F":
					numero_Ecuaciones_Adicionales++;
				break;
				// Incremento por CORTO CIRCUITO
				case "W":
					numero_Ecuaciones_Adicionales++;
				break;
				// Doble Incremento por Fuente de VOLTAJE controlada
				// por CORRIENTE
				case "H":
					numero_Ecuaciones_Adicionales++;
```

```
                                        numero_Ecuaciones_Adicionales++;
                                break;
                                // Incremento por Fuente de VOLTAJE controlada
                                // por VOLTAJE
                                case "E":
                                        numero_Ecuaciones_Adicionales++;
                                break;
                                // - Incremento por AMPLIFICADOR OPERACIONAL ---
                                case "A":
                                        numero_Ecuaciones_Adicionales++;
                                break;
                        }
                }

                // --- Inicializacion de la MATRIZ "A" y los vectores "b" y "xv" ---
                for (i=0; i<N.length+numero_Ecuaciones_Adicionales; i++) {
                        renglones = new Array();
                        for (j=0; j<N.length+numero_Ecuaciones_Adicionales; j++) {
                                renglones.push(new Point()); // valor-->(re,s)
                        }
                        // --- Creacion de la MATRIZ DE CONDUCTANCIAS con valor cero
                        A.push(renglones);
                        // --- Creacion del VECTOR DE VOLTAJES con valor cero ---
                        xv.push(0);
                        // --- Creacion del VECTOR DE CORRIENTES con valor cero ---
                        b.push(new Point());      // valor-->(re,s)
                }
        }

        private function inicializar_Matriz_y_Vectores():void {

                // --- Auxiliares ------------
                var id:String;
                // --- Nodos de conexion -----
                var n1:int;
                var n2:int;
                // --- Nodos controladores ---
                var nc1:int;
                var nc2:int;
                // --- Valor de componente ---
                var valor:Number;
                // --- Tipo de fuente de voltaje ---
                var tipo:String;
                // --- Contadores ------------
                numero_Ecuaciones_Adicionales = 0;

                // --- Inicializacion de la Matriz "A y los vectores
                // --- "b" y zvect
                for (i=0; i<C.length; i++) {
                        id = C[i].id;
                        n1 = C[i].c[0];
                        n2 = C[i].c[1];
                        valor = C[i].valor;
                        if (C[i].tipo != undefined) {
                                tipo = C[i].tipo;
                        }
```

```
                                        //V_0 = C[i].xxx

                                        switch(id.substr(0,1)) {
                                                // --- Resistencia ----------------------------------
                                                case "R":
                                                        A[n1][n1].x += 1/valor;
                                                        A[n2][n2].x += 1/valor;
                                                        A[n2][n1].x -= 1/valor;
                                                        A[n1][n2].x -= 1/valor;
                                                break;
                                                // --- Capacitor ------------------------------------
                                                case "C":
                                                        A[n1][n1].y += valor;
                                                        A[n2][n2].y += valor;
                                                        A[n2][n1].y -= valor;
                                                        A[n1][n2].y -= valor;
                                                break;
                                                // --- Fuente de Voltaje independiente  ---------------
                                                case "V":
                                                        numero_Ecuaciones_Adicionales++;
                                                        A[n1][A[0].length-
numero_Ecuaciones_Adicionales].x = 1;
                                                        A[n2][A[0].length-
numero_Ecuaciones_Adicionales].x = -1;
                                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n1].x = 1;
                                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n2].x = -1;
                                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =
"I_"+id;
                                                        if (tipo == analisis) {
                                                                // Analisis en AC
                                                                b[A[0].length-
numero_Ecuaciones_Adicionales].x = valor;
                                                        } else {
                                                                // Analisis en DC
                                                                // --- Pasivacion de fuentes de voltaje ---
                                                                b[A[0].length-
numero_Ecuaciones_Adicionales].x = 0;
                                                        }
                                                break;
                                                // --- Fuente de Corriente independiente  -------------
                                                case "I":
                                                        if (tipo == analisis) {
                                                                // Analisis en AC
                                                                b[n1].x += valor;
                                                                b[n2].x += -valor;
                                                        } esle {
                                                                // Analisis en DC
                                                                b[n1].x += 0;
                                                                b[n2].x += 0;
                                                        }
                                                break;
                                                // --- Fuente de CORRIENTE controlada por VOLTAJE "G"
                                                // --- FCCV ----
                                                case "G":
```

```
                                        nc1 = C[i].c[2];
                                        nc2 = C[i].c[3];
                                        A[n1][nc1].x += valor;    // +g
                                        A[n1][nc2].x += -valor;   // -g
                                        A[n2][nc1].x += -valor;   // -g
                                        A[n2][nc2].x += valor;    // +g
                                break;
                                // --- Fuente de CORRIENTE controlada por CORRIENTE
"F"
                                // --- FCCC ---
                                case "F":
                                        nc1 = C[i].c[2];
                                        nc2 = C[i].c[3];
                                        numero_Ecuaciones_Adicionales++;
                                        A[nc1][A[0].length-
numero_Ecuaciones_Adicionales].x = 1;
                                        A[nc2][A[0].length-
numero_Ecuaciones_Adicionales].x = -1;
                                        A[n1][A[0].length-
numero_Ecuaciones_Adicionales].x =

                                valor;  // Beta
                                        A[n2][A[0].length-
numero_Ecuaciones_Adicionales].x =

                                -valor; // -Beta
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][nc1].x = 1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][nc2].x = -1;
                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =

                                "I_"+id;
                                break;
                                // --- Fuente de VOLTAJE controlada por CORRIENTE "H"
                                // --- FVCC ---
                                case "H":
                                        nc1 = C[i].c[2];
                                        nc2 = C[i].c[3];
                                        numero_Ecuaciones_Adicionales++;
                                        A[n1][A[0].length-
numero_Ecuaciones_Adicionales].x = 1;
                                        A[n2][A[0].length-
numero_Ecuaciones_Adicionales].x = -1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n1].x = 1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n2].x = -1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][A[0].length-numero_Ecuaciones_Adicionales-1].x = -valor;    // -r
                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =
"Ir_"+id;
                                        numero_Ecuaciones_Adicionales++;
                                        A[nc1][A[0].length-
numero_Ecuaciones_Adicionales].x = 1;
```

```
                                        A[nc2][A[0].length-
numero_Ecuaciones_Adicionales].x = -1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][nc1].x = 1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][nc2].x = -1;
                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =
"I_"+id;
                                break;
                                // --- Fuente de VOLTAJE controlada por VOLTAJE "E"
                                // --- FVCV ---
                                case "E":
                                        nc1 = C[i].c[2];
                                        nc2 = C[i].c[3];
                                        numero_Ecuaciones_Adicionales++;
                                        A[n1][A[0].length-
numero_Ecuaciones_Adicionales].x = 1;
                                        A[n2][A[0].length-
numero_Ecuaciones_Adicionales].x = -1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][nc1].x = -valor;         // -u
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][nc2].x = valor;          // u
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n1].x = 1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n2].x = -1;
                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =
"Iu_"+id;
                                break;
                                // --- AMPLIFICADOR OERACIONAL "A" --- FVCV ---
                                case "A":          // n1(V+); n2(V-); nc1(Vo)
                                        nc1 = C[i].c[2];
                                        numero_Ecuaciones_Adicionales++;
                                        A[nc1][A[0].length-
numero_Ecuaciones_Adicionales].x = -1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n1].x = 1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n2].x = -1;
                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =
"I_"+id;
                                break;
                                // --- CORTO CIRCUITO ---------------------------------
                                case "W":
                                        numero_Ecuaciones_Adicionales++;
                                        A[n1][A[0].length-
numero_Ecuaciones_Adicionales].x = 1;
                                        A[n2][A[0].length-
numero_Ecuaciones_Adicionales].x = -1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n1].x = 1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][n2].x = -1;
                                        A[A[0].length-
numero_Ecuaciones_Adicionales][A[0].length-numero_Ecuaciones_Adicionales].x = valor;
```

```
                                        xv[A[0].length-numero_Ecuaciones_Adicionales] =
"I_"+id;
                                break;
                        }
                }

                // Cortar renglon "0" de "A", "xv" y "b", correspondiente
                // al nodo de "tierra"
                A.splice(0,1);
                xv.splice(0,1);
                b.splice(0,1);
                // --- Actualiza el vector "xv" ---
                for (i=0; i<N.length-1; i++) {
                        xv[i] = "V"+(i+1);
                }
                // Cortar columna "0" de "A" , correspondiente
                // al nodo de "tierra"
                for (i=0; i<A.length; i++) {
                        A[i].splice(0,1);
                }
        }

        // ------------------------------------------------------
        //
        //                      Seccion de impresion de resultados:
        //                                      |A||xv| = |b|
        //
        // ------------------------------------------------------

        public function imprime_Matriz_y_Vectores():void {

                trace(" ");
                trace("          *** MATRIZ ORIGINAL COMPLEJA [A] ***");
                for (i=0; i<A.length; i++) {
                        trace("-----------------------------------------------------");
                        trace("    A   \t re   s");
                        for (j=0; j<A[i].length; j++) {
                                trace(" A["+i+"]["+j+"]:\t"+A[i][j]);
                        }
                }

                trace(" ");
                trace("*** VECTOR [xv] ***");
                for (i=0; i<xv.length; i++) {
                        trace(" "+xv[i]);
                }

                trace(" ");
                trace("********** VECTOR [b] ********************");
                trace(" b.length= "+b.length);
                trace("    b\t re   s");
                for (i=0; i<b.length; i++) {
                        trace(" b["+i+"]:\t"+b[i]);
                }
        }
```

```
		public function imprime_Matriz_y_Vectores_FormatoMatricial():void {

			var renglonX:Array;
			var renglonY:Array;
			trace(" ");

			trace(" ");
			trace("::  FORMATO MATRICIAL  *** MATRIZ ORIGINAL COMPLEJA [A]
***");
			for (i=0; i<A.length; i++) {
				renglonX = new Array();
				renglonY = new Array();
				for (j=0; j<A[i].length; j++) {
					renglonX.push(A[i][j].x);
					renglonY.push(A[i][j].y);
				}
				trace(" ");
				trace("re: "+i+" "+renglonX);
				trace("im: "+i+" "+renglonY);
			}

			trace(" ");
			trace("*** VECTOR [xv] ***");
			for (i=0; i<xv.length; i++) {
				trace(" "+xv[i]);
			}

			trace(" ");
			trace("********** VECTOR [b] **********");
			trace(" b.length= "+b.length);
			trace("    b\t re   s");
			for (i=0; i<b.length; i++) {
				trace(" b["+i+"]:\t"+b[i]);
			}
		}
	}
}


package com.red.simulador.ac
{
	// ************************************************************
	//
	//	Clase:  Metodo_Gauss_Jordan_Complejo
	//
	//	Patron de Programacion:
	//
	//				Auxiliar de la clase "simuladorAC"
	//
	//                                                                    septiembre
/ 2010
	// ************************************************************
	//
	//	ENTRADA:	Matriz compleja aumentada "A" y vectores "xv" y "b"
	//
	//                                                          |A||xv| = |b|
```

```
		//                                          con:
		//                                                  |A|  --> Point: Matriz compleja aumentada
		//                                                  |xv| --> String: Vector de variables a
calcular
		//                                                                  (voltajes nodales y
corrientes de malla)
		//                                                  |b|  --> Point: Vector complejo de excitacion
		//
		//      SALIDA:             |xv| = |b = solucion|
		//                                          con:
		//                                                  |xv| --> String: Vector de variables a
calcular
		//                                                                  (voltajes nodales y
corrientes de malla)
		//                                                  |b|  --> Point: Vector solucion
		//
		//                                  - Servicios de Impresion
		//                                          * imprime_Solucion_Matriz_Ab()
		//
		// ************************************************************

		// Matematicas
		import flash.geom.Point;

		 public class Metodo_Gauss_Jordan_Complejo {

			// --- Analisis_Nodal_Modificado_Complejo ---
			private var sistema:Analisis_Nodal_Modificado_Complejo;
			// --- Tipo de analisis ---
			public var analisis:String;

			// Voltajes Nodales + Corrientes de:
			// Fuente Independiente de Voltaje, Alambre o Capacitor
			public var Voltajes_Nodales_Extendido:Array;

			private var OA:Array;	// Matriz Original
			private var A:Array;	// Matriz de trabajo

			public var xv:Array;	// Variables a calcular
			private var b:Array;	// Vector de trabajo
			private var Ob:Array;	// Vector Original

			// --- Indices ---
			private var i:int;
			private var j:int;
			private var k:int;

			// --- Auxiliares ---
			private var permutaciones:int;

			// --- Variable compleja ---
			private var s:Point;

			// --- Matematicas ---
			private var operacion_compleja:Operaciones_Numeros_Complejos;
```

```
		// --- Frecuencia especifica ---
		private var frecuencia:Number;
		private var jw:Number;

		public function Metodo_Gauss_Jordan_Complejo(

_sistema:Analisis_Nodal_Modificado_Complejo):void {

			// --- Recepcion de la matriz y los vectores ---
			sistema = _sistema;

			// --- Operaciones complejas auxiliares ----------
			operacion_compleja = new Operaciones_Numeros_Complejos();
		}

		public function calcula_Voltajes_Corrientes_AC(
							frecuencia:Number):Object {

			// Objeto para guardado de la solucion
			// del sistema de ecuaciones
			var obj:Object = new Object();

			// --- Tipo de analisis ---
			analisis = sistema.analisis;

			// --- Frecuencia (Hz) especifica de analisis ---
			this.frecuencia = frecuencia;
			// Frecuencia compleja (rad/seg) como componente
			// de la variable compleja "s = jw"
			jw = (Math.round(2*Math.PI*frecuencia*100))/100;
			// --- Variable compleja "s = jw"
			s = new Point(0,jw);

			// Arreglo para el guardado de la solucion del
			// sistema de ecuaciones complejas
			Voltajes_Nodales_Extendido = new Array();

			// --- Creacion de la matriz compleja |A| ---
			crear_Matriz_A();
			// --- Solucion de la matriz compeja |A| ---
			solucion_Matriz_Ab(s);
			// --- Guardado de las soluciones ---
			Voltajes_Nodales_Extendido = b;

			obj.xv = xv;
			obj.Voltajes_Nodales_Extendido = Voltajes_Nodales_Extendido;

			return obj;
		}

		private function crear_Matriz_A():void {

			// --- Matriz |A| y vector |b| de trabajo ---
			A = new Array();
			b = new Array();
```

```
		// Matriz |OA| y vectores |Ob| y |xv| originales
		// Contienen la descripcion matricial compleja
		// del circuito que se va simular
		OA = sistema.A;
		xv = sistema.xv;
		Ob = sistema.b;

		// --- Creacion de la matriz de trabajo |A| ---
		for (i=0; i<OA.length; i++) {
			A[i] = new Array();
			for (j=0; j<OA[i].length; j++) {
				A[i][j] = new Point();
			}
		}
}

public function solucion_Matriz_Ab(_s:Point):void {

		// ------------------------------------------------------------
		//		Solucion de la matriz Ab por el metodo de Gauss Jordan
		// ------------------------------------------------------------

		// --- Auxiliares ---
		var pivote:Number;
		var elemento:Number;
		var suma:Point;
		var signo:Number;
		var r:Point;
		var temp1:Point;
		var temp2:Point;
		var s:Number;
		var renglon:int;
		var m:int;

		permutaciones = 0;

		// ------------------------------------------------------------
		//				Evaluacion de la matriz |A| y el vector |b|
		//					con el valor de "s = jw"
		// ------------------------------------------------------------
		for (i=0; i<A.length; i++) {
			for (j=0; j<A[i].length; j++) {
				A[i][j] = new Point();
				A[i][j] = A_evaluacion_AC(OA[i][j],_s);
			}
			b[i] = new Point();
			b[i] = Ob[i];
		}

		// --- Creacion de la MATRIZ TRIANGULAR SUPERIOR ---
		// --- Proceso DIRECTO ---
		for (i=0; i<A.length; i++) {
			renglon = i;
			pivote = mayor(A[i][i]);
			j = i+1;
```

```
                        // --- Busqueda del mayor elemento de la matriz ---
                        for (j=i+1; j<A.length; j++) {
                                elemento = mayor(A[j][i]);
                                if (pivote < elemento) {
                                        pivote = elemento;
                                        renglon = j;
                                }
                        }

                        // --- Decision para intercambio de renglones --
                        if (renglon != i) {
                                for (j=i; j<A.length; j++) {
                                        // --- Intercambio en la matriz "A" ---
                                        temp1 = A[renglon][j];
                                        A[renglon][j] = A[i][j];
                                        A[i][j] = temp1;
                                }
                                // --- Intercambio en el vector "b" ---
                                temp1 = b[renglon];
                                b[renglon] = b[i];
                                b[i] = temp1;
                                // --- Registra numero de permutaciones ---
                                permutaciones++;
                        }

                        // --- Actualizacion del pivote ---
                        pivote = mayor(A[i][i]);

                        // --- Deteccion de matriz singular ---
                        if (pivote == 0) {
                                trace("*********************************");
                                trace("*   ERROR: Matriz singular    *");
                                trace("*              <PROCESO DIRECTO>                    *");
                           trace("         pivote = 0          *");
                                trace("*********************************");
                                break;
                        } else {
                                // --- Calculo de los elementos de la nueva matriz "A" ---
                                for (j=i+1; j<A.length; j++) {
                                        r = operacion_compleja.cociente(A[j][i],A[i][i]);
                                        for (k=i; k<A[i].length; k++) {
                                                temp1 =
operacion_compleja.producto(r,A[i][k]);
                                                temp2 =
operacion_compleja.resta(A[j][k],temp1);
                                                A[j][k] = temp2;
                                        }
                                        temp1 = operacion_compleja.producto(r,b[i]);
                                        temp2 = operacion_compleja.resta(b[j],temp1);
                                        b[j] = temp2;
                                }
                        }
                }

                // --- Primer paso de la sustitucion inversa ---
                b[A.length-1] = operacion_compleja.cociente(b[A.length-1],
```

```
                                                                                A[A.length-
1][A.length-1]);
                        // --- Continuacion del proceso de sustitucion inversa ---
                        for (i=1; i<A.length; i++) {
                                suma = new Point();
                                j = A.length-1-i;
                                k = j+1;
                                for (m=k; m<A.length; m++) {
                                        suma = operacion_compleja.suma(

suma,operacion_compleja.producto(A[j][m],b[m]));
                                }
                                b[j] = operacion_compleja.cociente(operacion_compleja.resta(

b[j],suma),A[j][j]);
                                // ------------------------------------------------------------
                                //   Ajusta valores de parte real e imaginaria a CERO
                                // si son menores que la referencia especificada en la
                                //                      CLASE: "Operaciones_Numeros_Complejos"
                                //                                  referencia = 1.0e-12
                                // ------------------------------------------------------------
                                operacion_compleja.ajusta_VALOR(b[j]);
                        }
                        operacion_compleja.ajusta_VALOR(b[A.length-1]);
                }

                private function A_evaluacion_AC(dato:Point,_s:Point):Point {
                        // ------------------------------------------------------------
                        //                          Evaluacion de la matriz |A| en el punto
                        //                           "s=jw" especificado: Point("Re", "Im")
                        // ------------------------------------------------------------
                        var aux:Point = operacion_compleja.productoEscalar(_s,dato.y);
                        return new Point(aux.x + dato.x,aux.y);
                }

                private function mayor(a_jb:Point):Number {
                        // ------------------------------------------------------------
                        //                  Determina el componente mayor ("Re" o "Im")
                        //                                  de un numero complejo
                        // ------------------------------------------------------------
                        var n_mayor:Number;
                        if(Math.abs(a_jb.x) >= Math.abs(a_jb.y)) {
                                n_mayor = Math.abs(a_jb.x);
                        } else {
                                n_mayor = Math.abs(a_jb.y);
                        }
                        return n_mayor;
                }

                public function imprime_Solucion_Matriz_Ab():void {

                        // ------------------------------------------------------------
                        //          |xv| = |b = solucion|
                        //                  con:
                        // |xv| --> Vector de variables a calcular
                        //                          (voltajes nodales y corrientes de malla)
```

```
                        // Voltajes_Nodales_Extendido[i].x --> Parte "real"
                        //
de la solucion compleja
                        // Voltajes_Nodales_Extendido[i].y --> Parte "imaginaria"
                        //
de la solucion compleja
                        // ----------------------------------------------------------
                        trace(" ");
                        trace("************************************************************");
                        trace(" VOLTAJES NODALES EXTENDIDO CALCULADOS A "+
                                                        frecuencia+"Hz, jw= "+jw+" Rad/s");
                        trace("************************************************************");
                        trace(" ");
                        trace(" Variable Magnitud          Fase");
                        for (i=0; i<Voltajes_Nodales_Extendido.length; i++) {
                                trace(" "+xv[i]+" = "+Voltajes_Nodales_Extendido[i].x+
                                                " + "+Voltajes_Nodales_Extendido[i].y+"j");
                        }
                }

        }
}

package com.red.simulador.ac
{
        // ************************************************************
        //
        //      Clase:  Operaciones_Numeros_Complejos
        //
        //      Patron de Programacion:
        //
        //              Auxiliar de la clase "Analisis_Nodal_Modificado_Complejo"
        //
        //                                                      marzo / 2010
        // ************************************************************
        //
        //      ENTRADA:    a, b  --> Point: donde "a" y "b" son numeros complejos
        //                          n     --> Int o Number
        //
        //      SALIDA:             suma(a:Point,b:Point) --> Point
        //                          resta(a:Point,b:Point) --> Point
        //                          productoEscalar(a:Point,b:Number) --> Point
        //                  producto(a:Point,b:Point) --> Point
        //                          conjugado(a:Point) --> Point
        //                          cocienteEscalar(a:Point,b:Number) --> Point
        //                          cociente(a:Point,b:Point) --> Point
        //                          cocientePolar(a:Point,b:Point) --> Point
        //                          modulo(a:Point) --> Number
        //                          fase(a:Point) --> Number
        //                          convierte_a_Polar(a:Point) --> Point
        //                          compara_CERO(a:Point) --> Boolean
        //                          ajusta_VALOR(a:Point)
        //
        //                          generar_Puntos_Equidistantes(n:int) --> Array
        //                                                  (en el circulo unitario: "exp(jw)")
        //                          potencia(a:Point,n:Number) --> Point
```

```
//                                  exponencialCompleja(n:Number) --> Point
//                                                          (exp(n) = cos(2*pi*n) + jsen(2*pi*n))
//
// ************************************************************

// Matematicas
import flash.geom.Point;

 public class Operaciones_Numeros_Complejos {

        private var referencia:Number = 1.0e-12;
        private var redondeo:Number = 1.0e7;

        public function Operaciones_Numeros_Complejos():void {
                // CONSTRUCTOR
        }

        public function suma(a:Point,b:Point):Point {
                //return a.add(b);
                return new Point(a.x+b.x,a.y+b.y);
        }

        public function resta(a:Point,b:Point):Point {
                return a.subtract(b);
        }

        public function productoEscalar(a:Point,b:Number):Point {
                return new Point(a.x*b,a.y*b);
        }

        public function producto(a:Point,b:Point):Point {
                return new Point((a.x*b.x)-(a.y*b.y), (a.x*b.y)+(a.y*b.x));
        }

        public function conjugado(a:Point):Point {
                return new Point(a.x,-a.y);
        }

        public function cocienteEscalar(a:Point,b:Number):Point {
                return new Point(a.x/b, a.y/b);
        }

        public function cociente(a:Point,b:Point):Point {
                return cocienteEscalar(producto(a,conjugado(b)),
                                                        producto(b,conjugado(b)).x);
        }

        public function cocientePolar(a:Point,b:Point):Point {
                return new Point(a.x/b.x,a.y-b.y);
        }

        public function modulo(a:Point):Number {
                return Math.abs(Math.sqrt(a.x*a.x+a.y*a.y));
        }

        public function fase(a:Point):Number {
```

```
		var fase_real:Number;

		if ((a.y != 0) && (a.x != 0)) {
			fase_real = 180*(Math.atan2(a.y,a.x))/Math.PI;
		} else if ((a.y == 0) && (a.x > 0)) {
			fase_real = 0;
		} else if ((a.y == 0) && (a.x < 0)) {
			fase_real = -180;
		} else if ((a.y == 0) && (a.x == 0)) {
			fase_real = 0;
		} else if ((a.y > 0) && (a.x == 0)) {
			fase_real = 90;
		} else if ((a.y < 0) && (a.x == 0)) {
			fase_real = -90;
		}

		return fase_real;
	}

	public function convierte_a_Polar(a:Point):Point {
		return new Point(modulo(a),fase(a));
	}

	public function compara_CERO(a:Point):Boolean {
		if (modulo(a) < referencia) {
			return true;
		} else {
			return false;
		}
	}

	public function ajusta_VALOR(a:Point):void {
		if ((Math.abs(a.x) < referencia) &&
			(Math.abs(a.y) < referencia)) {
			a.x = 0;
			a.y = 0;
		} else if (Math.abs(a.x) < referencia) {
			a.x = 0
		} else if (Math.abs(a.y) < referencia) {
			a.y = 0
		}

		if (Math.abs(a.x)) {
			a.x = Math.round(a.x*redondeo)/redondeo;
		}
		if (Math.abs(a.y)) {
			a.y = Math.round(a.y*redondeo)/redondeo;
		}
	}

	public function generar_Puntos_Equidistantes(n:int):Array {

		var s:Array = new Array();
		var k:int;
		var re:Number;
```

```
                var im:Number;
                var temp:Point;

                for (k=0; k<=n; k++) {
                        re = Math.cos((2*k*Math.PI)/(n+1));
                        im = Math.sin((2*k*Math.PI)/(n+1));
                        temp = new Point(re,im);
                        //ajusta_VALOR(temp);
                        s.push(temp);
                }

                return s;
        }

        public function potencia(a:Point,n:Number):Point {

                var k:int;
                var temp:Point = a;
                var nn:Number = Math.abs(n);

                switch(nn) {
                        case 0:
                                temp = new Point(1,0);
                        break;
                        case 1:
                                temp = a;
                        break;
                        default:
                                for (k=0; k<nn-1; k++) {
                                        temp = producto(temp,a);
                                }
                        break;
                }

                if (n<0) {
                        temp = cociente(new Point(1,0),temp);
                }

                return temp;
        }

        public function exponencialCompleja(n:Number):Point {

                var re:Number;
                var im:Number;
                var temp:Point;

                re = Math.cos(2*n*Math.PI);
                im = Math.sin(2*n*Math.PI);
                temp = new Point(re,im);
                ajusta_VALOR(temp);

                return temp;
        }
    }
}
```

```
package Vistas.dc
{
	// ****************************************************************
	//
	//		Clase:	VistasDC
	//
	//		Patrones de Programacion:
	//
	//		(1) "MVC" (MODELO - VISTA - CONTROLADOR)
	//
	//		(2) "COMPOSICION" (motor3D_dc <-- vistasDC --> graficaBarras)
	//                                                                                                  |
	//                                                                                                  V

	//                                                                                          listaDatos
	//
	//                                                                                                  septiembre /
2010
	// ****************************************************************

	// Flash principal
	import flash.display.Sprite;

	// Campos de texto para etiquetas
	import flash.text.TextField;
	import flash.text.TextFieldAutoSize;
	import flash.text.TextFieldType;
	// Eventos
	import flash.events.MouseEvent;
	import flash.events.Event;

	// Representaciones en 2D y 3D
	import Vistas.dc.graficas.graficaBarras;
	import Vistas.dc.listas.listaDatos;
	import Vistas.dc.animacion3D.motor3D_dc;

	// Modelo MVC
	import Modelo.ModeloCircuito;

	//import Vistas.dc.graficas.graficaLineas;
	//import com.bin.ventana;

	public class vistasDC extends Sprite {

		// --- Parametros recibidos ---
		private var spEditor:Sprite;
		private var nodosVisuales:Array;
		private var elementosCircuito:Array;

		// --- Representaciones en 2D y 3D ---
		private var barrasVoltaje:graficaBarras;
		private var listaComponentes:listaDatos;
		private var circuito3D:motor3D_dc;

		// --- Modelo MVC ---
		public var modelo:ModeloCircuito;

		public function vistasDC (modelo:ModeloCircuito) {
			this.modelo = modelo;
		}

		// ----------------------------------------------------------
		//
		// Mostrar resultados en "grafica de barras" y "circuito en 3D"
		//
		// ----------------------------------------------------------

		public function mostrarDatos(spEditor:Sprite,nodosVisuales:Array):void
		{
			this.spEditor = spEditor;
```

```
                this.nodosVisuales = nodosVisuales;

                // --- Captura referencias a los componentes del circuito ---
                elementosCircuito = new Array();
                for (i=0; i<spEditor.numChildren; i++) {
                        elementosCircuito.push(spEditor.getChildAt(i));
                }

                // Captura "alturas" y "voltajes nodales" de los
                // "nodosVisuales"
                var voltajesNodales:Array = new Array();
                for (i in nodosVisuales) {
                        voltajesNodales.push([nodosVisuales[i].z_n,
                                                                        nodosVisuales[i].voltajeNodal]);
                }

                // --- Lista de componentes ---
                listaComponentes = new listaDatos(elementosCircuito);
                listaComponentes.x = 450;
                listaComponentes.y = 60;
                addChild (listaComponentes);

                // --- Grafica de barras para voltajes nodales ---
                barrasVoltaje = new graficaBarras(this,voltajesNodales);
                addChild (barrasVoltaje);
                barrasVoltaje.x = 480;
                barrasVoltaje.y = 400;

                // --- Grafica del circuito en 3D ---
                circuito3D = new motor3D_dc(spEditor,nodosVisuales);
                addChild(circuito3D);
        }

        // ------------------------------------------------------------
        //
        //                      Liberacion de objetos para regresar al EDITOR
        //
        // ------------------------------------------------------------

        public function liberarDatos():void {
                circuito3D.removerElementos();
                removeChild (circuito3D);
                removeChild (barrasVoltaje);
                removeChild (listaComponentes);
        }

        // ------------------------------------------------------------
        //
        //                      Conexion entre "graficaBarras" --> "motor3D"
        //
        // ------------------------------------------------------------

        public function seleccionarNodo(id:String):void {
                // --- Conexion con "motor3D" ---
                circuito3D.seleccionarNodo(id);
        }

        public function quitarSeleccion(id:String):void {
                // --- Conexion con "motor3D" ---
                circuito3D.quitarSeleccion(id);
        }

        public function despliegaNodosTerminales(voltajesNodales:Array):void {
                circuito3D.despliegaNodosTerminales(voltajesNodales);
        }
    }
}
```

```
package Vistas.dc.animacion3D
{
	// ****************************************************************
	//
	//		Clase:	motor3D_dc
	//
	//		Patron de Programacion:
	//
	//			"COMPOSICION" (motor3D_dc <-- vistasDC --> graficaBarras)
	//                                                                                  |
	//                                                                                  V
	//                                                                          listaDatos
	//
	//                                                                              septiembre /
2010
	// ****************************************************************
	//
	//		ENTRADA:		- Componentes del circuito en "spEditor" (Object)
	//                                  - Nodos visuales en "nodosVisuales" (Array)
	//
	//		CONEXIONES INTERACTIVAS:
	//                                  - Con "graficaBarras" a traves de "vistasDC"
	//
	//		SALIDA:
	//                                  - Circuito en 3D en pantalla
	//                                  - Voltajes nodales en DC (altura de nodos)
	//                                  - Corrientes de rama en DC (flechas)
	//
	// ****************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Sliders
	import fl.controls.Slider;
	import fl.events.SliderEvent;
	// Campos de texto para etiquetas
	import flash.text.*;
	// Matematicas
	import flash.geom.*;

	public class motor3D_dc extends Sprite {

		// --- Parametros recibidos ---
		public var spEditor:Object;
		public var componente:Object;
		private var elementosCircuito:Array;
		private var nodosVisuales:Array;

		// --- Plano de referencia y "linea de potencial positivo" ---
		private var referencia:MovieClip;
		private var ladoPlanoReferencia:Number = 200;

		// --- Referencias a "Componentes" y "Nodos Visuales" ---
		public var vectorElementos3D:Vector.<MovieClip>;
		// --- Coordenadas 3D de los "Componentes" y "Nodos Visuales" ---
		public var vectorCoordenadas3D:Vector.<Vector3D>;

		// Cubo virtual contenedor de los
		// "Componentes" y "Nodos Visuales"
		public var cuboContenedorVirtual:Sprite;

		// --- Auxiliares para control de rotacion del cubo virtual ---
		public var estaRotandoCubo:Boolean;
		public var prevX:Number;
		public var prevY:Number;

		// --- Contenedor del cubo virtual ---
```

```
		public var container:Sprite;

		// --- Sliders ---
		public var sliders:Array;
		private var xSlider:Slider;
		private var ySlider:Slider;
		private var zSlider:Slider;
		// --- Auxiliares para Sliders ---
		public var prevXVal:Number;
		public var prevYVal:Number;
		public var prevZVal:Number;

		// Auxiliar para separar las referencias a los
		// "Componentes" y los "Nodos Visuales"
		private var offset:uint;

		public function motor3D_dc(spEditor:Sprite,nodosVisuales:Array):void {

			// --- Recepcion de parametros ---
			this.spEditor = spEditor;
			this.container = spEditor;
			this.nodosVisuales = nodosVisuales;

			// --- Inicializacion de Vectores ---
			vectorElementos3D = new Vector.<MovieClip>(spEditor.numChildren +
nodosVisuales.length+1);
			vectorCoordenadas3D = new Vector.<Vector3D>(spEditor.numChildren +
nodosVisuales.length+1);

			// --- Coordenadas en 3D ---
			var x_coord:Number;
			var y_coord:Number;
			var z_coord:Number;

			construirMotor3D();
		}

		internal function construirMotor3D():void {

			// --- Inicializar elementos 3D ---
			inicializarElementos3D();
			construirPlanoReferencia();

			// --- Inicializar cubo virtual 3D ---
			inicializarCuboVirtual();
			construirCuboVirtual();
			inicializarListenersCubo();

			// --- Crear Sliders ---
			crearSliders();
			inicializarListenersSliders();

			// Funcion de clasificacion y ordenamiento de la distancia,
			// en 3D, de los elementos visuales al observador
			clasificarElementos3DZ();
		}

		// ------------------------------------------------------------
		//
		//                                          Inicializacion de los vectores:
		//                    "vectorElementos3D" y "vectorCoordenadas3D"
		//
		// ------------------------------------------------------------

		internal function inicializarElementos3D():void {

			// ------------------------------------------------------------
			// Traslado de elementos de los "Componentes" y los "Nodos Visuales" a los
			// vectores "vectorElementos3D" y sus coordenadas 3D a los
			// vectores "vectorCoordenadas3D", para poder manejarlos en
```

```
		// espacio 3D
		// ------------------------------------------------------------
		for (i=0; i<spEditor.numChildren; i++) {
			// --- TrasladoPlanoReferencia de los "Componentes"  ---
			vectorElementos3D[i] = spEditor.getChildAt(i);
			componente = spEditor.getChildAt(i);
			// --- Asignacion de las coordenadas 3D respectivas ---
			x_coord = componente.datosComponente[0][2];
			y_coord = componente.datosComponente[0][3];
			z_coord = componente.datosComponente[0][4];
			vectorCoordenadas3D[i] = new Vector3D(x_coord,y_coord,z_coord);
		}

		offset = container.numChildren;
		for (i=offset; i<offset + nodosVisuales.length; i++) {
			// --- TrasladoPlanoReferencia de los "Nodos Visuales"  ---
			vectorElementos3D[i] = nodosVisuales[i-offset];
			// --- Asignacion de las coordenadas 3D respectivas ---
			nodo = vectorElementos3D[i];
			x_coord = nodo.x_n;
			y_coord = nodo.y_n;
			z_coord = nodo.z_n;
			nodo.x = x_coord;
			nodo.y = y_coord;
			nodo.z = -z_coord;
			vectorCoordenadas3D[i] = new Vector3D(x_coord,y_coord,z_coord);
		}
}

// ------------------------------------------------------------
//
//		Inicializacion del "Plano de Referencia de potencial 0"
//
// ------------------------------------------------------------

internal function construirPlanoReferencia():void {

		// --- Construccion del plano de referencia a potencial 0 volts ---
		referencia = new MovieClip();
		// --- Plano ---
		var plano:Sprite = new Sprite();
		plano.graphics.lineStyle(1,0x00ccff,0.5);
		plano.graphics.beginFill(0x0055ff,0.2);
		plano.graphics.moveTo(-ladoPlanoReferencia,-ladoPlanoReferencia);
		plano.graphics.lineTo(ladoPlanoReferencia,-ladoPlanoReferencia);
		plano.graphics.lineTo(ladoPlanoReferencia,ladoPlanoReferencia);
		plano.graphics.lineTo(-ladoPlanoReferencia,ladoPlanoReferencia);
		plano.graphics.lineTo(-ladoPlanoReferencia,-ladoPlanoReferencia);
		plano.graphics.endFill();
		referencia.addChild(plano);
		// --- Linea indicadora de potencial positivo ---
		var lineaReferenciaPositiva:Sprite = new Sprite();
		lineaReferenciaPositiva.graphics.lineStyle(1,0x00ccff,0.8);
		lineaReferenciaPositiva.graphics.moveTo(0,0);
		lineaReferenciaPositiva.graphics.lineTo(100,0);
		lineaReferenciaPositiva.rotationY = 90;
		referencia.addChild(lineaReferenciaPositiva);
		// --- Registro del plano de referencia en el "vectorElementos3D" ---
		vectorElementos3D[vectorElementos3D.length-1] = referencia;
		// --- Registro de las coordendas del plano de referencia  ---
		vectorCoordenadas3D[vectorCoordenadas3D.length-1] = new Vector3D(0,0,0);
}

// ------------------------------------------------------------
//
//		Seccion de "Cubo Virtual", el cual contiene a todos los
//			elementos visbles que se presentan en 3D
//
// ------------------------------------------------------------
```

```
public function inicializarCuboVirtual():void {

		// --- Construye cubo virtual como contenedor para animacion 3D ---
		cuboContenedorVirtual = new Sprite();
		cuboContenedorVirtual.name = cuboContenedorVirtual;
		container.addChild(cuboContenedorVirtual);

		cuboContenedorVirtual.x=0;
		cuboContenedorVirtual.y=0;
		cuboContenedorVirtual.z=0;
		// --- Flag ---
		estaRotandoCubo = false;
}

public function construirCuboVirtual() {

		// ------------------------------------------------------------
		// Todos los elementos visuales (componentes y nodos visuales)
		// se trasladan del editor (spEditor) al "cuboContenedorVirtual"
		// para representarse en 3D
		// ------------------------------------------------------------
		var i:int;
		for (i=0; i<vectorElementos3D.length; i++) {
			cuboContenedorVirtual.addChild(vectorElementos3D[i]);
		}
}

public function inicializarListenersCubo():void {
		container.addEventListener(MouseEvent.ROLL_OUT,detenerMovimientoCubo);
		container.addEventListener(MouseEvent.MOUSE_MOVE,moverCuboVirtual);
		container.addEventListener(MouseEvent.MOUSE_DOWN,prepararCuboVirtual);
		container.addEventListener(MouseEvent.MOUSE_UP,detenerMovimientoCubo);
}

public function prepararCuboVirtual(e:MouseEvent):void {
		prevX=container.mouseX;
		prevY=container.mouseY;
		estaRotandoCubo=true;
}

public function detenerMovimientoCubo(e:MouseEvent):void {
		estaRotandoCubo=false;
}

public function moverCuboVirtual(e:MouseEvent):void {
		var locX:Number=prevX;
		var locY:Number=prevY;
		if(estaRotandoCubo){
			prevX=container.mouseX;
			prevY=container.mouseY;
			rotarCuboVirtual(prevY-locY,-(prevX-locX),0);
			e.updateAfterEvent();
		}
}

public function rotarCuboVirtual(rotx:Number,roty:Number,rotz:Number):void {
		cuboContenedorVirtual.transform.matrix3D.appendRotation(rotx,Vector3D.X_AXIS);
		cuboContenedorVirtual.transform.matrix3D.appendRotation(roty,Vector3D.Y_AXIS);
		cuboContenedorVirtual.transform.matrix3D.appendRotation(rotz,Vector3D.Z_AXIS);
		clasificarElementos3DZ();
}

// ------------------------------------------------------------
//
//                                                  Funciones "auxiliares"
//
// ------------------------------------------------------------

public function clasificarElementos3DZ():void {
```

```
                        // ----------------------------------------------------------
                        // Funcion de clasificacion y ordenamiento de la distancia,
                        // en 3D, de los elementos visuales al observador
                        // ----------------------------------------------------------
                        var arregloDistanciasElementos:Array=new Array();
                        var i:int;
                        var matriz3DActual:Matrix3D;
                        var puntosMediosElementos:Vector3D;
                        matriz3DActual = cuboContenedorVirtual.transform.matrix3D.clone();

                        // ----------------------------------------------------------
                        // Preparacion de los elementos visuales contenidos en el
                        // vector "puntosMediosElementos" en el arreglo
                        // "arregloDistanciasElementos" para su clasificacion
                        // ----------------------------------------------------------
                        for(i=0; i<vectorCoordenadas3D.length; i++){

puntosMediosElementos=matriz3DActual.deltaTransformVector(vectorCoordenadas3D[i]);
                                arregloDistanciasElementos[i]={ distance:puntosMediosElementos.z,

which:i};
                        }

                        // ----------------------------------------------------------
                        // Clasificacion de las "distancias" al observador de cada
                        // elemento contenido en el "cuboContenedorVirtual"
                        // ----------------------------------------------------------
                        arregloDistanciasElementos.sortOn("distance", Array.NUMERIC  |

              Array.DESCENDING);
                        // ----------------------------------------------------------
                        // Limpiar el "cuboContenedorVirtual" de los elementos
                        // actuales
                        // ----------------------------------------------------------
                        while(cuboContenedorVirtual.numChildren>0){
                                cuboContenedorVirtual.removeChildAt(0);
                        }

                        // ----------------------------------------------------------
                        // Llenar el "cuboContenedorVirtual" con los elementos ya
                        // clasificados por su distancia al observador, de acuerod a
                        // la posicion actual del "cuboContenedorVirtual" despues de
                        // haberlo rotado
                        // ----------------------------------------------------------
                        for(i=0; i<vectorElementos3D.length; i++){

cuboContenedorVirtual.addChild(vectorElementos3D[arregloDistanciasElementos[i].which]);
                        }
                }

                public function removerElementos():void {

                        // ----------------------------------------------------------
                        // Remover elementos y listeners del "cuboContenedorVirtual"
                        // antes de regresar al estado "editando"
                        // ----------------------------------------------------------
                        container.removeChild(cuboContenedorVirtual);
                        container.removeEventListener(MouseEvent.ROLL_OUT,detenerMovimientoCubo);
                        container.removeEventListener(MouseEvent.MOUSE_MOVE,moverCuboVirtual);
                        container.removeEventListener(MouseEvent.MOUSE_DOWN,prepararCuboVirtual);
                        container.removeEventListener(MouseEvent.MOUSE_UP,detenerMovimientoCubo);
                        // --- Remover Sliders ---
                        xSlider.removeEventListener(SliderEvent.THUMB_DRAG,cambioSliderX);
                        ySlider.removeEventListener(SliderEvent.THUMB_DRAG,cambioSliderY);
                        zSlider.removeEventListener(SliderEvent.THUMB_DRAG,cambioSliderZ);
                        removeChild(xSlider);
                        removeChild(ySlider);
                        removeChild(zSlider);
                }
```

```
// ------------------------------------------------------------
//
//                    Conexion entre "graficaBarras" --> "motor3D"
//          (activacion de la animacion de los "nodosVisuales" desde
//                                                    la "graficaBarras")
//
// ------------------------------------------------------------

public function seleccionarNodo(id:String):void {
        for (i in nodosVisuales) {
                if (id == nodosVisuales[i].name) {
                        nodosVisuales[i].iniciarAnimacion();
                }
        }
}

public function quitarSeleccion(id:String):void {
        for (i in nodosVisuales) {
                if (id == nodosVisuales[i].name) {
                        nodosVisuales[i].detenerAnimacion();
                }
        }
}

// ------------------------------------------------------------
//
//                                                    Seccion de "Sliders"
//
// ------------------------------------------------------------

private function crearSliders():void {

        // --- Creacion y colocacion de "xSlider" ---
        xSlider = new Slider();
        xSlider.setSize(180,10);
        xSlider.maximum = 180;
        xSlider.minimum = -180;
        xSlider.x = 780;
        xSlider.y = 300;
        addChild(xSlider);
        // --- Creacion y colocacion de la etiqueta para "xSlider" ---
        var xLabel:mc_label_SliderX = new mc_label_SliderX();
        xLabel.x = xSlider.x + 78;
        xLabel.y = xSlider.y + 9;
        addChild(xLabel);

        // --- Creacion y colocacion de "ySlider" ---
        ySlider = new Slider();
        ySlider.setSize(180,10);
        ySlider.maximum = 180;
        ySlider.minimum = -180;
        ySlider.x = 780;
        ySlider.y = 340;
        addChild(ySlider);
        // --- Creacion y colocacion de la etiqueta para "ySlider" ---
        var yLabel:mc_label_SliderY = new mc_label_SliderY();
        yLabel.x = ySlider.x + 78;
        yLabel.y = ySlider.y + 9;
        addChild(yLabel);

        // --- Creacion y colocacion de "zSlider" ---
        zSlider = new Slider();
        zSlider.setSize(180,10);
        zSlider.maximum = 180;
        zSlider.minimum = -180;
        zSlider.x = 780;
        zSlider.y = 380;
        addChild(zSlider);
        // --- Creacion y colocacion de la etiqueta para "zSlider" ---
        var zLabel:mc_label_SliderZ = new mc_label_SliderZ();
```

```
                zLabel.x = zSlider.x + 78;
                zLabel.y = zSlider.y + 9;
                addChild(zLabel);

                // --- Inicializacion de variables de Sliders ---
                prevXVal = 0;
                prevYVal = 0;
                prevZVal = 0;
            }

            internal function inicializarListenersSliders():void {
                xSlider.addEventListener(SliderEvent.THUMB_DRAG,cambioSliderX);
                ySlider.addEventListener(SliderEvent.THUMB_DRAG,cambioSliderY);
                zSlider.addEventListener(SliderEvent.THUMB_DRAG,cambioSliderZ);
            }

            function cambioSliderX(e:SliderEvent):void {
                var curXVal:Number=e.target.value;
                rotarCuboVirtual(curXVal-prevXVal,0,0);
                prevXVal=curXVal;
            }

            function cambioSliderY(e:SliderEvent):void {
                var curYVal:Number=e.target.value;
                rotarCuboVirtual(0,curYVal-prevYVal,0);
                prevYVal=curYVal;
            }

            function cambioSliderZ(e:SliderEvent):void {
                var curZVal:Number=e.target.value;
                rotarCuboVirtual(0,0,curZVal-prevZVal);
                prevZVal=curZVal;
            }
        }
}
```

```
package Vistas.dc.graficas
{
	// ****************************************************************
	//
	//		Clase:	graficaBarras
	//
	//		Patron de Programacion:
	//
	//				"COMPOSICION" (motor3D_dc <-- vistasDC --> graficaBarras)
	//																|
	//																V

	//															listaDatos
	//
	//																septiembre /
2010
	// ****************************************************************
	//
	//		ENTRADA:		- Voltajes nodales en "arregloDatos" (Array)
	//						Estructura de "arregloDatos"
	//							- arregloDatos[0][0] --> voltaje nodal "normalizado"
	//																	en
"modeloCircuito"
	//							- arregloDatos[0][1] --> voltaje nodal calculado
	//																	por
el simulador
	//
	//		CONEXIONES INTERACTIVAS:
	//						- Con "motor3D_dc" a traves de "vistasDC"
	//
	//		SALIDA:
	//						- Grafica de barras de voltajes nodales
	//
	// ****************************************************************

	// Flash principal
	import flash.display.*;
	// Campos de texto para etiquetas
	import flash.text.*;
	// Eventos
	import flash.events.*;

	// Vistas
	import Vistas.vistasDC;

	public class graficaBarras extends Sprite {

		// --- Parametros recibidos ---
		private var arregloDatos:Array;
		private var vistas:Object;

		// --- Variables para graficacion de datos ---
		private var b:Array;
		private var maximo:Number=0;
		private var indiceMaximo:Number=0;
		private var referenciaEjeY:Number;
		private var rangoDinamico:uint =160;
		private var anchoBarra:Number;
		private var alturaBarra:Number;

		// --- Parametros para definicion de grafica ---
		private var anchoGrafica:Number = 210;
		private var altoGrafica:Number = rangoDinamico+20;
		private var deltaAcotacion:Number = 20;		//(altoGrafica-10)/10;
		private var posicionX:Number;

		// --- Acotaciones y etiquetas ---
		private var acotacionesY:Array=[];
		private var etiquetasX:Array=[];
		private var etiquetasY:Array=[];
```

```
private var valorVoltaje:TextField;
private var etiquetaNodo:TextField;
private var linea:Sprite;

// --- Contenedor ---
private var contenedor:Sprite;

public function graficaBarras (vistas:Object,datos:Array) {

	// --- Recepcion de datos ---
	this.vistas = vistas;
	this.arregloDatos = datos;

	// --- Creacion de contenedor de la grafica ---
	contenedor = new Sprite();
	addChild(contenedor);
	linea = new Sprite();

	contruirGraficaBarras();
}

private function contruirGraficaBarras():void {
	inicializarGraficaBarras();
	dibujarFondoGrafica();
	dibujarAcotacionesY();
	dibujarEtiquetasX();
	dibujarEtiquetasY();
	dibujarGraficaBarras();
}

// ------------------------------------------------------------
//
//                                        Seccion de Grafica de Barras
//
// ------------------------------------------------------------

private function inicializarGraficaBarras():void {

	// Selecciona maximo valor de los
	// voltajes nodales normalizados
	for (var i in arregloDatos) {
		if (Math.abs(arregloDatos[i][0]) > maximo) {
			maximo = Math.abs(arregloDatos[i][0]);
			indiceMaximo = i;
		}
	}

	// --- Normaliza altura de barras ---
	alturaBarra = rangoDinamico/2;
	b = new Array();
	for (i in arregloDatos) {
		if (maximo != 0) {
			b.push((arregloDatos[i][0]/maximo)*alturaBarra);
		} else {
			b.push(arregloDatos[i][0]*alturaBarra);
		}
	}
	// --- Calcula ancho de cada barra ---
	anchoBarra = 200/b.length - 2;
	// --- Calcula posicion de cada barra ---
	posicionX = anchoBarra + 2;
	// --- Inicializa referencia de voltaje CERO ---
	referenciaEjeY = -altoGrafica/2;
}

public function dibujarFondoGrafica():void {
	var fondo:Shape = new Shape();
	fondo.graphics.lineStyle(1,0xffffff);
	fondo.graphics.beginFill(0x000000);
	fondo.graphics.drawRect(-5,0,anchoGrafica,-altoGrafica);
```

```
		fondo.graphics.endFill();
		contenedor.addChild(fondo);
}

public function dibujarAcotacionesY():void {

		var acotaY:Shape;
		for (dy=10; dy<altoGrafica; dy += deltaAcotacion) {
			acotaY = new Shape();
			if (dy == altoGrafica/2) {
				acotaY.graphics.lineStyle(1,0xffffff,1);
			} else {
				acotaY.graphics.lineStyle(1,0xffffff,0.3);
			}
			acotaY.graphics.moveTo(0,0);
			acotaY.graphics.lineTo(anchoGrafica,0);
			acotaY.x = -5;
			acotaY.y = -dy;
			acotacionesY.push(acotaY);
			contenedor.addChild(acotaY);
		}
}

private function dibujarEtiquetasX():void {

		// --- Titulo de la Grafica --
		var tituloGrafica:TextField = new TextField();
		tituloGrafica.text = "   Grafica de Voltajes Nodales ";
		tituloGrafica.x = 0;
		tituloGrafica.y = -altoGrafica - 20;
		tituloGrafica.width = 250;
		tituloGrafica.height = 25;

		// --- Formato de Texto ---
		var format0:TextFormat = new TextFormat();
		format0.font = "_sans";
		format0.size = 12;
		format0.color = 0xffffff;
		tituloGrafica.setTextFormat(format0);

		contenedor.addChild (tituloGrafica);

		for (var i=0; i<arregloDatos.length; i++) {
			etiquetasX[i]= new TextField();
			etiquetasX[i].text = "V"+i;
			etiquetasX[i].x=posicionX*i-4+anchoBarra/2;
			etiquetasX[i].y=0;
			etiquetasX[i].width = 20;
			etiquetasX[i].height = 15;

			// --- Formato de Texto ---
			var format:TextFormat = new TextFormat();
			format.font = "_sans";
			format.size = 10;
			format.color = 0xffffff;
			etiquetasX[i].setTextFormat(format);

			contenedor.addChild (etiquetasX[i]);
		}
}

private function dibujarEtiquetasY():void {

		var valorAcotacionMinima:Number = Math.abs(arregloDatos[indiceMaximo][1]);
		var deltaValorAcotacion:Number = valorAcotacionMinima/4;

		for (var i=0; i<acotacionesY.length; i++)
		{
			etiquetasY[i]= new TextField();
			etiquetasY[i].text = (Math.round((-valorAcotacionMinima+i*
```

```
                                                                    deltaValorAcotacion)*10)/10).toString()+"
V";
                                        etiquetasY[i].x = -40;
                                        etiquetasY[i].y = acotacionesY[i].y - 8;
                                        etiquetasY[i].width = 35;
                                        etiquetasY[i].height = 15;

                                        // --- Formato de Texto ---
                                        var format:TextFormat = new TextFormat();
                                        format.font = "_sans";
                                        format.size = 10;
                                        format.color = 0xffffff;
                                        etiquetasY[i].setTextFormat(format);

                                        contenedor.addChild (etiquetasY[i]);
                                }
                        }

                        private function dibujarGraficaBarras():void {

                                // --- Dibujo de la grafica de barras ---
                                var grafica:Sprite = new Sprite();

                                var bar:Sprite;
                                var alturaBarra:Number;

                                for (i in b) {
                                        bar = new Sprite();
                                        bar.buttonMode = true;
                                        bar.name = "n"+i;
                                        bar.alpha = 0.7;
                                        // --- Agrega "Listeners" para grafica interactiva ---
                                        bar.addEventListener(MouseEvent.MOUSE_OVER, seleccionarNodo);
                                        bar.addEventListener(MouseEvent.MOUSE_OUT, quitarSeleccion);
                                        // --- Dibujo de la barra ---
                                        bar.graphics.lineStyle (0,0x000000);
                                        bar.graphics.beginFill (0x0099cc);
                                        if (b[i] != 0) {
                                                bar.graphics.lineStyle(0,0x0099cc);
                                                bar.graphics.beginFill(0x0099cc);
                                                alturaBarra = b[i];
                                                if (b[i] > 0) {
                                                        // --- Voltaje positivo ---
                                                        bar.graphics.drawRect(posicionX*i,(referenciaEjeY-b[i]-1),

                        anchoBarra,alturaBarra);
                                                } else {
                                                        // --- Voltaje negativo ---
                                                        bar.graphics.drawRect(posicionX*i,(referenciaEjeY-b[i]+1),

                        anchoBarra,alturaBarra);
                                                }
                                        } else {
                                                // Si el voltaje a graficar es CERO, dibuja un cuadro negro
                                                // como barra, para tener interactividad cuando el mouse
                                                // pase sobre dicho cuadro
                                                bar.graphics.lineStyle(0,0x000000);
                                                bar.graphics.beginFill(0x000000);
                                                alturaBarra = -30;
                                                bar.graphics.drawRect(posicionX*i,(referenciaEjeY-b[i]-1),

        anchoBarra,alturaBarra/2);
                                                bar.graphics.drawRect(posicionX*i,(referenciaEjeY-b[i]+1),

        anchoBarra,-alturaBarra/2);
                                        }

                                        bar.graphics.endFill();
                                        grafica.addChild(bar);
                                }
```

```
                contenedor.addChild (grafica);
        }

        // ------------------------------------------------------------
        //
        //                  Conexion entre "graficaBarras" --> "motor3D"
        //          (activacion de la animacion de los "nodosVisuales" desde
        //                                                  la "graficaBarras")
        //
        // ------------------------------------------------------------

        public function seleccionarNodo(evt:Event):void {

                vistas.seleccionarNodo(evt.target.name);
                evt.target.alpha = 1;
                mostrarValorVoltaje(evt.target.name);
        }

        public function quitarSeleccion(evt:Event):void {

                vistas.quitarSeleccion(evt.target.name);
                evt.target.alpha = 0.7;

                contenedor.removeChild(etiquetaNodo);
                contenedor.removeChild (valorVoltaje);
                contenedor.removeChild(linea);
        }

        private function mostrarValorVoltaje(indice:String):void {

                var numNodo:Number;
                numNodo = Number(indice.substr(1,2))

                // --- Valor del voltaje ---
                valorVoltaje=new TextField();
                valorVoltaje.text = (Math.round(arregloDatos[numNodo][1]*
                                                                                        100)/100).toString()+"
V";
                valorVoltaje.x=anchoGrafica + 10;
                valorVoltaje.y=referenciaEjeY-b[numNodo]-10;
                valorVoltaje.border = true;
                valorVoltaje.borderColor = 0xffffff;
                valorVoltaje.background = true;
                valorVoltaje.backgroundColor = 0x0099cc;
                valorVoltaje.width = 40;
                valorVoltaje.height = 15;

                // --- Formato de Texto ---
                var format:TextFormat = new TextFormat();
                format.font = "_sans";
                format.size = 10;
                format.color = 0xffffff;
                valorVoltaje.setTextFormat(format);

                linea.graphics.clear();
                linea.graphics.lineStyle(0.2,0xffffff,0.8);
                linea.graphics.moveTo(etiquetasX[numNodo].x+anchoBarra/2,
                                                                                        referenciaEjeY-
b[numNodo]);
                linea.graphics.lineTo(anchoGrafica + 10,referenciaEjeY-b[numNodo]);

                // --- Etiqueta del Nodo ---
                etiquetaNodo=new TextField();
                etiquetaNodo.text = "V"+numNodo;
                etiquetaNodo.x=etiquetasX[numNodo].x;
                etiquetaNodo.y=etiquetasX[numNodo].y;
                etiquetaNodo.border = true;
                etiquetaNodo.borderColor = 0xffffff;
                etiquetaNodo.background = true;
                etiquetaNodo.backgroundColor = 0x0099cc;
```

```
                    etiquetaNodo.width = 20;
                    etiquetaNodo.height = 15;

                    // --- Formato de Texto ---
                    var format2:TextFormat = new TextFormat();
                    format2.font = "_sans";
                    format2.size = 10;
                    format2.color = 0xffffff;
                    etiquetaNodo.setTextFormat(format2);

                    contenedor.addChild(linea);
                    contenedor.addChild(valorVoltaje);
                    contenedor.addChild(etiquetaNodo);
                }
            }
}
```

```
package Vistas.dc.listas
{
	// ************************************************************
	//
	//		Clase:	listasDatos
	//
	//		Patron de Programacion:
	//
	//			"COMPOSICION" (motor3D_dc <-- vistasDC --> graficaBarras)
	//                                                                                  |
	//                                                                                  V

	//                                                                          listaDatos
	//
	//                                                                                  septiembre
/ 2010
	// ************************************************************

	import flash.display.*;
	import flash.events.*;

	import fl.controls.*;
	import fl.controls.DataGrid;
	import fl.containers.*;
	import fl.core.UIComponent;
	import fl.data.DataProvider;

	public class listaDatos extends Sprite {

		public var dp:DataProvider;
		public var dg:DataGrid;

		private var arregloDatos:Array;
		public var info:Sprite;

		public function listaDatos(lista:Array):void {

			arregloDatos = lista;

			dp = new DataProvider();
			for (var indice in arregloDatos) {
				if (arregloDatos[indice].name.substr(0,1) != "W" &&
					arregloDatos[indice].name.substr(0,1) != "T") {
					dp.addItem( { label:
arregloDatos[indice].datosComponente[0], data:arregloDatos[indice].name} );
					//dp.addItem( { label:
"e"+(indice)+"="+arregloDatos[indice].datosComponente[0], data:arregloDatos[indice].name} );

	//listaUI.addItem({label:"e"+(indice)+"="+arregloDatos[indice].datosComponente[0],data:arre
gloDatos[indice].name});
				}
			}

			var cb:ComboBox = new ComboBox();
			cb.move(10,10);
			cb.setSize(200,25);
```

```
                        cb.prompt = "Select a component to view its info";
                        cb.rowCount = 12;
                        cb.dataProvider = dp;
                        cb.addEventListener(Event.CHANGE, showStyleDefinition);
                        addChild(cb);

                        info = new Sprite();
                        info.graphics.lineStyle(1,0x00ff99,0.8);
                        info.graphics.beginFill(0x22cc00,0.5);
                        info.graphics.drawRect(0,0,300,100);
                        info.graphics.endFill();
                        info.x = 20;
                        info.y = 100;
                        //addChild(info);

                        /*dg = new DataGrid();
                        //dg.setSize(425,300);
                        dg.setSize(170, 250);
                        dg.move(450,120);
                        dg.columns = [ "Name", "Goals" ];
                        //dg.columns = [ new DataGridColumn("StyleName"), new
DataGridColumn("DefaultValue") ];
                        dg.dataProvider = dp;
                        addChild(dg);*/

                }

                public function showStyleDefinition(e:Event):void {
                        trace(e.target);
                        trace(e.target.selectedItem.data);
                        addChild(info);
                        /*var componentClass:Class = e.target.selectedItem.data as Class;
                        var styles:Object = componentClass["getStyleDefinition"].call(this);
                        trace(styles.toString());
                        var styleData:DataProvider = new DataProvider();
                        for(var i:* in styles) {
                                trace(i + " : " + styles[i]);
                                styleData.addItem( { StyleName:i, DefaultValue:styles[i] } );
                        }
                        styleData.sortOn("StyleName");
                        dg.dataProvider = styleData;*/
                }
        }
}
```

```
package Vistas.ac
{
	// ************************************************************
	//
	//		Clase:   VistasAC
	//
	//		Patrones de Programacion:
	//
	//		(1) "MVC" (MODELO - VISTA - CONTROLADOR)
	//
	//		(2) "COMPOSICION" (motor3D_ac <-- vistasAC --> graficaBarras)
	//                                                                                  |
	//                                                                                  V
	//                          Graficador_Respuesta_Frecuencia)
	//
	//                                                                                septiembre
/ 2010
	// ************************************************************

	// Flash principal
	import flash.display.Sprite;

	// Campos de texto para etiquetas
	import flash.text.TextField;
	import flash.text.TextFieldAutoSize;
	import flash.text.TextFieldType;
	// Eventos
	import flash.events.MouseEvent;
	import flash.events.Event;

	// Representaciones en 2D y 3D
	import Vistas.dc.graficas.graficaBarras;
	import Vistas.ac.respuestaFrecuencia.Graficador_Respuesta_Frecuencia;
	import Vistas.ac.respuestaFrecuencia.Cursor_Respuesta_Frecuencia;
	import Vistas.ac.animacion3D.motor3D_ac;

	// Modelo MVC
	import Modelo.Generador_Lista_Datos;
	import Modelo.ModeloCircuito;

	public class vistasAC extends Sprite {

		// --- Parametros recibidos ---
		private var spEditor:Sprite;
		private var elementosCircuito:Array;
		private var nodosVisuales:Array;
		private var LD:Generador_Lista_Datos;

		// --- Representaciones en 2D y 3D ---
		private var graficaRF:Graficador_Respuesta_Frecuencia;
		private var cursor:Cursor_Respuesta_Frecuencia;
		private var barrasVoltaje:graficaBarras;
		private var circuito3D:motor3D_ac;

		// --- Modelo MVC ---
		public var modelo:ModeloCircuito;
```

```
		public function vistasAC (modelo:ModeloCircuito):void {
			this.modelo = modelo;
		}

		// ---------------------------------------------------------
		//
		// Mostrar la Respuesta en Frecuencia entre "f_min" y "f_max"
		//
		// ---------------------------------------------------------

		public function mostrarRespuestaFrecuencia(LD:Generador_Lista_Datos,

contenedor:Sprite):void {
			this.LD = LD;
			// --- Construccion de la grafica de Respuesta en Frecuencia ---
			graficaRF = new Graficador_Respuesta_Frecuencia(LD);
			graficaRF.graficarMagnituFase();
			addChild(graficaRF);

			// --- Colocacion del CURSOR interactivo ---
			cursor = new Cursor_Respuesta_Frecuencia(this,LD,graficaRF);
			cursor.inicializaCursorInteractivo();
			cursor.selecciona_Grafica_Nodo(0);
			addChild(cursor);
		}

// PENDIENTE:: Por desarrollar
public function actualiza_CoordX_CURSOR(coordenadaX:Number):void {
	trace("vistasAC lanzada desde el motor3D_AC:: coordenadaX= "+coordenadaX);
	//modelo.asignarResultadosSimulacionAC(coordenadaX);
}

		// ---------------------------------------------------------
		//
		//			Mostrar resultados en "grafica de barras",
		//		"circuito en 3D" y "Respuesta en Frecuencia"
		//
		// ---------------------------------------------------------

		public function mostrarDatos(spEditor:Sprite,nodosVisuales:Array):void
		{
			this.spEditor = spEditor;
			this.nodosVisuales = nodosVisuales;

			// --- Captura referencias a los componentes del circuito ---
			elementosCircuito = new Array();
			for (i=0; i<spEditor.numChildren; i++) {
				elementosCircuito.push(spEditor.getChildAt(i));
			}

			// Captura "magnitud" y "voltaje pico" de los
			// "nodosVisuales"
			var voltajesNodales:Array = new Array();
			for (i in nodosVisuales) {
```

```
                                voltajesNodales.push([-
nodosVisuales[i].magnitud,nodosVisuales[i].voltajeNodalPico]);
                        }

                        // --- Grafica de barras para magnitudes de voltajes nodales ---
                        barrasVoltaje = new graficaBarras(this,voltajesNodales);
                        addChild (barrasVoltaje);
                        barrasVoltaje.x = 480;
                        barrasVoltaje.y = 400;

                        // --- Grafica del circuito en 3D ---
                        circuito3D = new motor3D_ac(spEditor,nodosVisuales);
                        circuito3D.generaSeñalSenoidal();
                        addChild(circuito3D);
                }

                // ------------------------------------------------------------
                //
                //              Liberacion de objetos para regresar al EDITOR
                //
                // ------------------------------------------------------------

                public function liberarDatos():void {
                        circuito3D.removerElementos();
                        removeChild (circuito3D);
                        removeChild (barrasVoltaje);
                        removeChild(graficaRF);
                        removeChild(cursor);
                }

                // ------------------------------------------------------------
                //
                //      Conexion entre:
                //              * "graficaBarras" --> "motor3D"
                //              * "graficaBarras" --> "Graficador_Respuesta_Frecuencia"
                //
                // ------------------------------------------------------------

                public function seleccionarNodo(id:String):void {
                        // --- Conexion con "motor3D" ---
                        circuito3D.seleccionarNodo(id);
                        // --- Conexion con "Graficador_Respuesta_Frecuencia" ---
                        cursor.selecciona_Grafica_Nodo(id);
                        graficaRF.mostrarNodoSeleccionado(id);
                }

                public function quitarSeleccion(id:String):void {
                        // --- Conexion con "motor3D" ---
                        circuito3D.quitarSeleccion(id);
                }

        }
}
```

```
package Vistas.ac.animacion3D
{
	// ****************************************************************
	//
	//		Clase:	motor3D_ac
	//
	//		Patron de Programacion:
	//
	//			"COMPOSICION" (motor3D_ac <-- vistasAC --> graficaBarras)
	//                                                                                                  |
	//                                                                                                  V
	//                          Graficador_Respuesta_Frecuencia)
	//
	//                                                                                                  septiembre /
2010
	// ****************************************************************
	//
	//		ENTRADA:		- Componentes del circuito en "spEditor" (Object)
	//                                      - Nodos visuales en "nodosVisuales" (Array)
	//
	//		CONEXIONES INTERACTIVAS:
	//                                      - Con "graficaBarras" a traves de "vistasAC"
	//
	//		SALIDA:
	//                                      - Circuito en 3D en pantalla
	//                                      - Voltajes nodales complejos en AC
	//                                        magnitud y fase (movimiento en "z" de nodos)
	//
	// ****************************************************************

	// Flash principal
	import flash.display.*;
	// Eventos
	import flash.events.*;
	// Sliders
	import fl.controls.Slider;
	import fl.events.SliderEvent;
	// Campos de texto para etiquetas
	import flash.text.*;
	// Matematicas
	import flash.geom.*;

	public class motor3D_ac extends Sprite {

		// --- Parametros recibidos ---
		public var spEditor:Object;
		public var componente:Object;
		private var elementosCircuito:Array;
		private var nodosVisuales:Array;

		// --- Plano de referencia y "linea de potencial positivo" ---
		private var referencia:MovieClip;
		private var ladoPlanoReferencia:Number = 200;

		// --- Referencias a "Componentes" y "Nodos Visuales" ---
		public var vectorElementos3D:Vector.<MovieClip>;
		// --- Coordenadas 3D de los "Componentes" y "Nodos Visuales" ---
		public var vectorCoordenadas3D:Vector.<Vector3D>;

		// Cubo virtual contenedor de los
		// "Componentes" y "Nodos Visuales"
		public var cuboContenedorVirtual:Sprite;

		// --- Auxiliares para control de rotacion del cubo virtual ---
		public var estaRotandoCubo:Boolean;
		public var prevX:Number;
		public var prevY:Number;

		// --- Contenedor del cubo virtual ---
		public var container:Sprite;
```

```
		// --- Sliders ---
		public var sliders:Array;
		private var xSlider:Slider;
		private var ySlider:Slider;
		private var zSlider:Slider;
		// --- Auxiliares para Sliders ---
		public var prevXVal:Number;
		public var prevYVal:Number;
		public var prevZVal:Number;

		// Auxiliar para separar las referencias a los
		// "Componentes" y los "Nodos Visuales"
		private var offset:uint;

		// Variable independiente de tiempo para la generacion
		// de la funcion "cosenoidal" para la representacion
		// dinamica del circuito simulado
		private var t:Number = 1;

		public function motor3D_ac(spEditor:Sprite,nodosVisuales:Array):void {

			// --- Recepcion de parametros ---
			this.spEditor = spEditor;
			this.container = spEditor;
			this.nodosVisuales = nodosVisuales;

			// --- Inicializacion de Vectores ---
			vectorElementos3D = new Vector.<MovieClip>(spEditor.numChildren +
nodosVisuales.length+1);
			vectorCoordenadas3D = new Vector.<Vector3D>(spEditor.numChildren +
nodosVisuales.length+1);

			// --- Coordenadas en 3D ---
			var x_coord:Number;
			var y_coord:Number;
			var z_coord:Number;

			construirMotor3D();
		}

		internal function construirMotor3D():void {

			// --- Inicializar elementos 3D ---
			inicializarElementos3D();
			construirPlanoReferencia();

			// --- Inicializar cubo virtual 3D ---
			inicializarCuboVirtual();
			construirCuboVirtual();
			inicializarListenersCubo();

			// --- Crear Sliders ---
			crearSliders();
			inicializarListenersSliders();

			// Funcion de clasificacion y ordenamiento de la distancia,
			// en 3D, de los elementos visuales al observador
			clasificarElementos3DZ();
		}

		// ------------------------------------------------------------
		//
		//                                        Inicializacion de los vectores:
		//                    "vectorElementos3D" y "vectorCoordenadas3D"
		//
		// ------------------------------------------------------------

		internal function inicializarElementos3D():void {
```

```
                        // ---------------------------------------------------------
                        // Traslado de elementos de los "Componentes" y los "Nodos Visuales" a los
                        // vectores "vectorElementos3D" y sus coordenadas 3D a los
                        // vectores "vectorCoordenadas3D", para poder manejarlos en
                        // espacio 3D
                        // ---------------------------------------------------------
                        for (i=0; i<spEditor.numChildren; i++) {
                                // --- TrasladoPlanoReferencia de los "Componentes"  ---
                                vectorElementos3D[i] = spEditor.getChildAt(i);
                                componente = vectorElementos3D[i];
                                // --- Asignacion de las coordenadas 3D respectivas ---
                                x_coord = componente.datosComponente[0][2];
                                y_coord = componente.datosComponente[0][3];
                                z_coord = 0;            //componente.datosComponente[0][4];
                                vectorCoordenadas3D[i] = new Vector3D(x_coord,y_coord,z_coord);
                        }

                        offset = spEditor.numChildren;
                        for (i=offset; i<offset + nodosVisuales.length; i++) {
                                // --- TrasladoPlanoReferencia de los "Nodos Visuales"  ---
                                vectorElementos3D[i] = nodosVisuales[i-offset];
                                // --- Asignacion de las coordenadas 3D respectivas ---
                                nodo = vectorElementos3D[i];
                                x_coord = nodo.x_n;
                                y_coord = nodo.y_n;
                                z_coord = 0;
                                nodo.x = x_coord;
                                nodo.y = y_coord;
                                nodo.z = z_coord;
                                vectorCoordenadas3D[i] = new Vector3D(x_coord,y_coord,z_coord);
                        }
                }
// ---------------------------------------------------------
                //
                //      Inicializacion del "Plano de Referencia de potencial 0"
                //
                // ---------------------------------------------------------

                internal function construirPlanoReferencia():void {

                        // --- Construccion del plano de referencia a potencial 0 volts ---
                        referencia = new MovieClip();
                        referencia.name = "plano";
                        // --- Plano ---
                        var plano:Sprite = new Sprite();
                        plano.graphics.lineStyle(1,0x00ccff,0.5);
                        plano.graphics.beginFill(0x0055ff,0.2);
                        plano.graphics.moveTo(-ladoPlanoReferencia,-ladoPlanoReferencia);
                        plano.graphics.lineTo(ladoPlanoReferencia,-ladoPlanoReferencia);
                        plano.graphics.lineTo(ladoPlanoReferencia,ladoPlanoReferencia);
                        plano.graphics.lineTo(-ladoPlanoReferencia,ladoPlanoReferencia);
                        plano.graphics.lineTo(-ladoPlanoReferencia,-ladoPlanoReferencia);
                        plano.graphics.endFill();
                        referencia.addChild(plano);
                        // --- Linea indicadora de potencial positivo ---
                        var lineaReferenciaPositiva:Sprite = new Sprite();
                        lineaReferenciaPositiva.graphics.lineStyle(1,0x00ccff,0.8);
                        lineaReferenciaPositiva.graphics.moveTo(0,0);
                        lineaReferenciaPositiva.graphics.lineTo(100,0);
                        lineaReferenciaPositiva.rotationY = 90;
                        referencia.addChild(lineaReferenciaPositiva);
                        // --- Registro del plano de referencia en el "vectorElementos3D" ---
                        vectorElementos3D[vectorElementos3D.length-1] = referencia;
                        // --- Registro de las coordendas del plano de referencia  ---
                        vectorCoordenadas3D[vectorCoordenadas3D.length-1] = new Vector3D(0,0,0);
                }

                // ---------------------------------------------------------
                //
                //      Seccion de "Cubo Virtual", el cual contiene a todos los
```

```
//                      elementos visbles que se presentan en 3D
//
// ------------------------------------------------------------

public function inicializarCuboVirtual():void {

        // --- Construye cubo virtual como contenedor para animacion 3D ---
        cuboContenedorVirtual = new Sprite();
        cuboContenedorVirtual.name = cuboContenedorVirtual;
        container.addChild(cuboContenedorVirtual);

        cuboContenedorVirtual.x=0;
        cuboContenedorVirtual.y=0;
        cuboContenedorVirtual.z=0;
        // --- Flag ---
        estaRotandoCubo = false;
}

public function construirCuboVirtual() {

        // ------------------------------------------------------------
        // Todos los elementos visuales (componentes y nodos visuales)
        // se trasladan del editor (spEditor) al "cuboContenedorVirtual"
        // para representarse en 3D
        // ------------------------------------------------------------
        var i:int;
        for (i=0; i<vectorElementos3D.length; i++) {
                cuboContenedorVirtual.addChild(vectorElementos3D[i]);
        }
}

public function inicializarListenersCubo():void {
        container.addEventListener(MouseEvent.ROLL_OUT,detenerMovimientoCubo);
        container.addEventListener(MouseEvent.MOUSE_MOVE,moverCuboVirtual);
        container.addEventListener(MouseEvent.MOUSE_DOWN,prepararCuboVirtual);
        container.addEventListener(MouseEvent.MOUSE_UP,detenerMovimientoCubo);
}

public function prepararCuboVirtual(e:MouseEvent):void {
        prevX=container.mouseX;
        prevY=container.mouseY;
        estaRotandoCubo=true;
}

public function detenerMovimientoCubo(e:MouseEvent):void {
        estaRotandoCubo=false;
}

public function moverCuboVirtual(e:MouseEvent):void {
        var locX:Number=prevX;
        var locY:Number=prevY;
        if(estaRotandoCubo){
                prevX=container.mouseX;
                prevY=container.mouseY;
                rotarCuboVirtual(prevY-locY,-(prevX-locX),0);
                e.updateAfterEvent();
        }
}

public function rotarCuboVirtual(rotx:Number,roty:Number,rotz:Number):void {
        cuboContenedorVirtual.transform.matrix3D.appendRotation(rotx,Vector3D.X_AXIS);
        cuboContenedorVirtual.transform.matrix3D.appendRotation(roty,Vector3D.Y_AXIS);
        cuboContenedorVirtual.transform.matrix3D.appendRotation(rotz,Vector3D.Z_AXIS);
        clasificarElementos3DZ();
}

// ------------------------------------------------------------
//
//                                                  Funciones "auxiliares"
//
```

```
		// ----------------------------------------------------------
		public function clasificarElementos3DZ():void {

			// ----------------------------------------------------------
			// Funcion de clasificacion y ordenamiento de la distancia,
			// en 3D, de los elementos visuales al observador
			// ----------------------------------------------------------
			var arregloDistanciasElementos:Array=new Array();
			var i:int;
			var matriz3DActual:Matrix3D;
			var puntosMediosElementos:Vector3D;
			matriz3DActual = cuboContenedorVirtual.transform.matrix3D.clone();

			// ----------------------------------------------------------
			// Preparacion de los elementos visuales contenidos en el
			// vector "puntosMediosElementos" en el arreglo
			// "arregloDistanciasElementos" para su clasificacion
			// ----------------------------------------------------------
			for(i=0; i<vectorCoordenadas3D.length; i++){

puntosMediosElementos=matriz3DActual.deltaTransformVector(vectorCoordenadas3D[i]);
				arregloDistanciasElementos[i]={ distance:puntosMediosElementos.z,

which:i};
			}

			// ----------------------------------------------------------
			// Clasificacion de las "distancias" al observador de cada
			// elemento contenido en el "cuboContenedorVirtual"
			// ----------------------------------------------------------
			arregloDistanciasElementos.sortOn("distance", Array.NUMERIC  |

		 Array.DESCENDING);
			// ----------------------------------------------------------
			// Limpiar el "cuboContenedorVirtual" de los elementos
			// actuales
			// ----------------------------------------------------------
			while(cuboContenedorVirtual.numChildren>0){
				cuboContenedorVirtual.removeChildAt(0);
			}

			// ----------------------------------------------------------
			// Llenar el "cuboContenedorVirtual" con los elementos ya
			// clasificados por su distancia al observador, de acuerod a
			// la posicion actual del "cuboContenedorVirtual" despues de
			// haberlo rotado
			// ----------------------------------------------------------
			for(i=0; i<vectorElementos3D.length; i++){

cuboContenedorVirtual.addChild(vectorElementos3D[arregloDistanciasElementos[i].which]);
			}
		}

		public function removerElementos():void {

			// ----------------------------------------------------------
			// Remover elementos y listeners del "cuboContenedorVirtual"
			// antes de regresar al estado "editando"
			// ----------------------------------------------------------

			removeEventListener(Event.ENTER_FRAME, onEnterFrame);

			container.removeChild(cuboContenedorVirtual);
			container.removeEventListener(MouseEvent.ROLL_OUT,detenerMovimientoCubo);
			container.removeEventListener(MouseEvent.MOUSE_MOVE,moverCuboVirtual);
			container.removeEventListener(MouseEvent.MOUSE_DOWN,prepararCuboVirtual);
			container.removeEventListener(MouseEvent.MOUSE_UP,detenerMovimientoCubo);
			// --- Remover Sliders ---
			xSlider.removeEventListener(SliderEvent.THUMB_DRAG,cambioSliderX);
```

```
		ySlider.removeEventListener(SliderEvent.THUMB_DRAG,cambioSliderY);
		zSlider.removeEventListener(SliderEvent.THUMB_DRAG,cambioSliderZ);
		removeChild(xSlider);
		removeChild(ySlider);
		removeChild(zSlider);
}

// ----------------------------------------------------------
//
//			Conexion entre "graficaBarras" --> "motor3D"
//		(activacion de la animacion de los "nodosVisuales" desde
//								la "graficaBarras")
// ----------------------------------------------------------

public function seleccionarNodo(id:String):void {
		for (i in nodosVisuales) {
			if (id == nodosVisuales[i].name) {
				nodosVisuales[i].iniciarAnimacion();
			}
		}
}

public function quitarSeleccion(id:String):void {
		for (i in nodosVisuales) {
			if (id == nodosVisuales[i].name) {
				nodosVisuales[i].detenerAnimacion();
			}
		}
}

// ----------------------------------------------------------
//
//			Seccion de Generacion de Senial Senoidal
//
// ----------------------------------------------------------

public function generaSeñalSenoidal():void {
		addEventListener(Event.ENTER_FRAME, onEnterFrame);
}

public function onEnterFrame(event:Event):void {

		// --- Generacion de tiempo ---
		t++;
		if (t >= 62) {
			t=0;
		}
		// ----------------------------------------------------------
		// Asignacion dinamica de potenciales a los elementos
		// contenidos en el cubo virtual a traves de la funcion
		// "asignaPotencial(t)"
		// ----------------------------------------------------------
		for (i=0; i<cuboContenedorVirtual.numChildren; i++) {
			objeto = cuboContenedorVirtual.getChildAt(i);
			if (objeto.name.substr(0,1) != "p") {
				objeto.asignaPotencial(t);
			}
		}
}

// ----------------------------------------------------------
//
//							Seccion de "Sliders"
//
// ----------------------------------------------------------

private function crearSliders():void {

		// --- Creacion y colocacion de "xSlider" ---
		xSlider = new Slider();
```

```
                xSlider.setSize(180,10);
                xSlider.maximum = 180;
                xSlider.minimum = -180;
                xSlider.x = 780;
                xSlider.y = 300;
                addChild(xSlider);
                // --- Creacion y colocacion de la etiqueta para "xSlider" ---
                var xLabel:mc_label_SliderX = new mc_label_SliderX();
                xLabel.x = xSlider.x + 78;
                xLabel.y = xSlider.y + 9;
                addChild(xLabel);

                // --- Creacion y colocacion de "ySlider" ---
                ySlider = new Slider();
                ySlider.setSize(180,10);
                ySlider.maximum = 180;
                ySlider.minimum = -180;
                ySlider.x = 780;
                ySlider.y = 340;
                addChild(ySlider);
                // --- Creacion y colocacion de la etiqueta para "ySlider" ---
                var yLabel:mc_label_SliderY = new mc_label_SliderY();
                yLabel.x = ySlider.x + 78;
                yLabel.y = ySlider.y + 9;
                addChild(yLabel);

                // --- Creacion y colocacion de "zSlider" ---
                zSlider = new Slider();
                zSlider.setSize(180,10);
                zSlider.maximum = 180;
                zSlider.minimum = -180;
                zSlider.x = 780;
                zSlider.y = 380;
                addChild(zSlider);
                // --- Creacion y colocacion de la etiqueta para "zSlider" ---
                var zLabel:mc_label_SliderZ = new mc_label_SliderZ();
                zLabel.x = zSlider.x + 78;
                zLabel.y = zSlider.y + 9;
                addChild(zLabel);

                // --- Inicializacion de variables de Sliders ---
                prevXVal = 0;
                prevYVal = 0;
                prevZVal = 0;
        }

        internal function inicializarListenersSliders():void {
                xSlider.addEventListener(SliderEvent.THUMB_DRAG,cambioSliderX);
                ySlider.addEventListener(SliderEvent.THUMB_DRAG,cambioSliderY);
                zSlider.addEventListener(SliderEvent.THUMB_DRAG,cambioSliderZ);
        }

        function cambioSliderX(e:SliderEvent):void {
                var curXVal:Number=e.target.value;
                rotarCuboVirtual(curXVal-prevXVal,0,0);
                prevXVal=curXVal;
        }

        function cambioSliderY(e:SliderEvent):void {
                var curYVal:Number=e.target.value;
                rotarCuboVirtual(0,curYVal-prevYVal,0);
                prevYVal=curYVal;
        }

        function cambioSliderZ(e:SliderEvent):void {
                var curZVal:Number=e.target.value;
                rotarCuboVirtual(0,0,curZVal-prevZVal);
                prevZVal=curZVal;
        }
}
```

```
}
```

```
package Vistas.ac.respuestaFrecuencia
```

```
{
	// ****************************************************************
	//
	//		Clase:	Graficador_Respuesta_Frecuencia
	//
	//		Patron de Programacion:
	//
	//			"COMPOSICION" (motor3D_ac <-- vistasAC --> graficaBarras)
	//                                                                                                  |
	//                                                                                                  V
	//				Graficador_Respuesta_Frecuencia)
	//
	//                                                                                                  abril
/ 2010
	// ****************************************************************
	//
	//		ENTRADA:
	//		- LF --> Array: Valores de Frecuencia en variacion logaritmica
	//                                      (de "fmin" a "fmax")
	//		* LD[i][0][indiceFrecuencia] = dato.xv[i]:		String (identificador)
	//		* LD[i][1][indiceFrecuencia] = formaRectagular: Point(x, jy}
	//		* LD[i][2][indiceFrecuencia] = Fasor:			Point(MAGNITUD, FASE} (Polar)
	//		* LD[i][3][indiceFrecuencia] = Magnitud_dB:		Number
	//		* LD[i][4][indiceFrecuencia] = Fase_Grados:	Number
	//
	//		CONEXIONES INTERACTIVAS:
	//                                      - Con "graficaBarras" a traves de "vistasDC"
	//
	//		SALIDA:		Graficas de Respuesta en Frecuencia de cada variable
	//                                      calculada por el simulador a cada frecuencia del barrido
	//                                      de frecuencias contenido en el archivo LF
	//
	// ****************************************************************

	// Flash principal
	import flash.display.*;
	// Campos de texto para etiquetas
	import flash.text.*;
	// Eventos
	import flash.events.*;
	// Matematicas
	import flash.geom.Point;

	// Generador_Lista_Datos
	import Modelo.Generador_Lista_Datos;

	// Operaciones con numeros complejos
	import com.red.simulador.ac.Operaciones_Numeros_Complejos;

	public class Graficador_Respuesta_Frecuencia extends Sprite{

		// --- Referencia al objeto "LD" ---
		private var LD:Generador_Lista_Datos

		// --- Datos contenidos en el objeto "LD" ---
		public var numero_Decadas_Frecuencia:Number;
		public var numero_Muestras:Number;
		private var voltajesMagnituddB:Array;
		private var voltajesFaseGrados:Array;
		public var funcion_a_Graficar:Array;

		// --- Graficas de Magnitud y Fase ---
		public var graficas_Frecuencia:Array;
		public var g_Magnitud:Sprite;
		public var g_Fase:Sprite;
		private var posXMag:Number = 60;
		private var posYMag:Number = 500;
		private var posXFase:Number = 550;
		private var posYFase:Number = 500;
		private var flagInicio:Boolean = true;
```

```
// --- Arreglos de valores maximos y minimos
//                de los valores a graficar
private var max_min_MagnituddB:Array;
private var max_min_FaseGrados:Array;

// --- Variables para formato de las graficas ---
public var posicion_Etiqueta_Frecuencia:Array;
public var altura_Graficas_MagFase:Number = 100;
public var ancho_Graficas_MagFase:Number;

// --- Indices ---
private var i:int;
private var j:int;
private var k:int;

// --- "Sprite" de referencia para graficar ---
public var contenedor:Sprite;

// --- Colores ---
public var color_Background:Number;
public var color_Fondo_Graficas:Number;
public var color_Lineas_Acotaciones:Number;
public var color_Letras_Acotaciones:Number;
public var color_Grafica:Number;

// --- Frecuencias maxima y minima para barrido de frecuencia ---
// Valor maximo de frecuencia
public var f_max:Number;
// Valor minimo de frecuencia
public var f_min:Number;
public var numero_decadas:Number;
// "alfa" de rejilla de graficas
private var alfa:Number = 0.3;

// --- Parametros para graficacion de la Magnitud en dB ---
// Array auxiliar de datos de graficas por nodo
public var datos_Graficacion:Array;
// delta de separacion en acotamiento magnitud
public var delta_dB:Number = 20;
// Maximo valor en dB de acotamiento de la Magnitud
public var MAX_ACOTAMIENTO_dB:Number = 200;
// Minimo valor en dB de acotamiento de la Magnitud
public var MIN_ACOTAMIENTO_dB:Number = -200;

// --- Parametros para graficacion de la Fase en Grados ---
public var grafica_Phase:Array = new Array();
// delta de separacion en acotamiento Fase
public var delta_Grados:Number = 90;
// Maximo valor en grados de acotamiento de la Fase
public var MAX_ACOTAMIENTO_FASE:Number = 180;
// Minimo valor en grados de acotamiento de la Fase
public var MIN_ACOTAMIENTO_FASE:Number = -180;

// --- Constante de conversion de logaritmo Natural a logaritmo Decimal ---
private var LN_LOG:Number = 0.434294;

// --- Operaciones con numeros complejos ---
private var operacion_compleja:Operaciones_Numeros_Complejos;

public function Graficador_Respuesta_Frecuencia(_LD:Generador_Lista_Datos):void {

        // --- Referencia al objeto "LD" (Lista de Datos) -------
        LD = _LD;

        // --- Inicializacion de parametros ---
        numero_Decadas_Frecuencia = LD.numero_Decadas_Frecuencia;
        numero_Muestras = LD.numero_Muestras;
        ancho_Graficas_MagFase = numero_Muestras;
        f_max = LD.f_max;
```

```
                f_min = LD.f_min;

                // --- Propiedades de color por default ---
                color_Background = 0x000000;                    // Negro
                color_Fondo_Graficas = 0x000000;            // Negro
                color_Lineas_Acotaciones = 0xffffff;    // Blanco
                color_Letras_Acotaciones = 0xffffff;    // Blanco
                color_Grafica = 0xff9900;                       // Naranja

                // --- Arreglo de graficas de Magnitud y Fase para cada Nodo ---
                graficas_Frecuencia = new Array();
        }

        public function graficarMagnituFase() {
                // --- Graficacion ---
                inicializa_Graficas_MagFase();
                graficar_Magnitud_Fase();
                // --- Grafica de Respuesta en Frecuencia del nodo de tierra ---
                mostrarNodoSeleccionado("n0");
        }

        // ------------------------------------------------------------
        //
        //          Seccion de preparacion de graficas de la Respuesta en
        //                                      Frecuencia (magnitud o fase)
        //
        // ------------------------------------------------------------

        private function inicializa_Graficas_MagFase():void {

                var g_Titulos:Sprite

                // ------------------------------------------------------------
                //                                                  Grafica de Magnitud
                // ------------------------------------------------------------
                g_Magnitud = new Sprite();
                g_Magnitud.x = posXMag;
                g_Magnitud.y = posYMag;
                addChild(g_Magnitud);

                // --- Capa 1: Fondo
                g_Magnitud.addChild(dibuja_Fondo_Grafica());

                // --- Capa 2: Acotamiento de Frecuencia
                g_Magnitud.addChild(dibuja_Acotamiento_Frecuencia());

                // --- Capa 3: Textos> Titulo, TituloY, TituloX y TituloNodo
                g_Magnitud.addChild(dibuja_Titulos_Grafica("Magnitud","dB",

"Frecuencia (Hz)"));


                // ------------------------------------------------------------
                //                                                  Grafica de Fase
                // ------------------------------------------------------------
                g_Fase = new Sprite();
                g_Fase.x = posXFase;
                g_Fase.y = posYFase;
                addChild(g_Fase);

                // --- Capa 1: Fondo
                g_Fase.addChild(dibuja_Fondo_Grafica());

                // --- Capa 2: Acotamiento de Frecuencia
                g_Fase.addChild(dibuja_Acotamiento_Frecuencia());

                // --- Capa 3: Textos> Titulo, TituloY, TituloX y TituloNodo
                g_Fase.addChild(dibuja_Titulos_Grafica("Fase","Grados",

"Frecuencia (Hz)"));
```

```
}

private function graficar_Magnitud_Fase():void {

		var nodo:int;
		var magnitud:int = 3;
		var fase:int = 4;
		var obj:Object;

		// ------------------------------------------------------------
		// Prepara todas las graficas de cada nodo del circuito en su
		// "sprite" correspondiente y se guardan en el arreglo
		// "graficas_Frecuencia" para su posterior acceso a traves
		// de la clase "Cursor_Respuesta_Frecuencia"
		// ------------------------------------------------------------
		// --- Funciones a graficar: "magnitudDB" o "faseGrados"
		funcion_a_Graficar = new Array();

		// --- Inicializa nodos 0 al "n" ---
		for (nodo=0; nodo<LD.LD.length; nodo++) {
			funcion_a_Graficar[nodo] = new Array();
			// [nodo][0] - magnitudDB
			funcion_a_Graficar[nodo][0] = new Array();
			// [nodo][1] - faseGrados
			funcion_a_Graficar[nodo][1] = new Array();
			// --- Prepara objeto con informacion en diferentes formatos ---
			obj = new Object();
			obj.nodo = nodo;
			obj.Magnitud_dB = draw_Magnitud_dB(nodo,magnitud);
			obj.datos_Grafica_Magnitud = datos_Graficacion;
			obj.Fase_Grados = draw_Fase_Grados(nodo,fase);
			obj.datos_Grafica_Fase = datos_Graficacion;
			// --- Guarda informacion en el arreglo "graficas_Frecuencia"
			graficas_Frecuencia.push(obj);
		}
}

private function draw_Magnitud_dB(nodo:int,tipo:int):Sprite {

		// Determina los valores maximo y minimo
		// de la Magnitud en dB
		voltajesMagnituddB = LD.LD[nodo][tipo];
		funcion_a_Graficar[nodo][0] = voltajesMagnituddB;
		max_min_MagnituddB = calcula_Maximo_Minimo(voltajesMagnituddB);

		// --- Construccion del arreglo de argumentos "arg" ----
		var arg:Array = new Array();
		arg.push(MIN_ACOTAMIENTO_dB);
		arg.push(MAX_ACOTAMIENTO_dB);
		arg.push(delta_dB);
		arg.push(max_min_MagnituddB);
		arg.push(nodo);
		arg.push(tipo);
		// Guarda la grafica de Respuesta en Frecuencia
		// "magnitud en dB" correspondiente al nodo
		var magnitud_dB:Sprite = new Sprite();
		magnitud_dB.addChild(draw_Funcion(arg));

		return magnitud_dB;
}

private function draw_Fase_Grados(nodo:int,tipo:int):Sprite {

		// Determina los valores maximo y minimo
		// de la Fase en Grados
		voltajesFaseGrados = LD.LD[nodo][tipo];
		funcion_a_Graficar[nodo][1] = voltajesFaseGrados;
		max_min_FaseGrados = calcula_Maximo_Minimo(voltajesFaseGrados);

		// --- Construccion del arreglo de argumentos "arg" ----
```

```
                var arg:Array = new Array();
                arg.push(MIN_ACOTAMIENTO_FASE);
                arg.push(MAX_ACOTAMIENTO_FASE);
                arg.push(delta_Grados);
                arg.push(max_min_FaseGrados);
                arg.push(nodo);
                arg.push(tipo);
                // Guarda la grafica de Respuesta en Frecuencia
                // "fase en Grados" correspondiente al nodo
                var fase_Grados:Sprite = new Sprite();
                fase_Grados.addChild(draw_Funcion(arg));

                return fase_Grados;
        }

        private function dibuja_Fondo_Grafica():Sprite {
                var sp:Sprite = new Sprite();
                sp.graphics.lineStyle(1,0xffffff,.8);
                sp.graphics.beginFill(color_Fondo_Graficas);
                sp.graphics.drawRect(0,0,numero_Muestras,altura_Graficas_MagFase);
                sp.graphics.endFill();
                return sp;
        }

        // ------------------------------------------------------------
        //
        //          Seccion de titulos y acotamientos de las graficas de
        //                      Respuesta en Frecuencia (magnitud o fase)
        //
        // ------------------------------------------------------------

        private function dibuja_Acotamiento_Frecuencia():Sprite {

                var sp:Sprite = new Sprite();

                var frecuencia:Number;
                var valor_Etiqueta_Frecuencia:Array = new Array();

                // --- Calcula posiciones de etiquetas ---
                posicion_Etiqueta_Frecuencia = new Array();
                posicion_Etiqueta_Frecuencia.push(0);

                // --- Grafica lineas verticales ---
                sp.graphics.lineStyle(1,color_Lineas_Acotaciones,alfa);
                for (k=0; k<numero_Decadas_Frecuencia; k++) {
                        for (i=1; i<=10; i++) {
                                // ------------------------------------------------------
                                // Genera 10 valores de frecuencia espaciados en forma
                                // logaritmica, con k = numero de decada:
                                //
                                //                                                  numero_Muestras
                                // frecuencia = -------------------------* (log10(i) + k)
                                //                                          numero_Decadas_Frecuencia
                                //
                                // ------------------------------------------------------
                                frecuencia = (numero_Muestras/numero_Decadas_Frecuencia)*

(LN_LOG*Math.log(i) + k);
                                sp.graphics.moveTo(frecuencia,0);
                                sp.graphics.lineTo(frecuencia,altura_Graficas_MagFase);
                        }
                        posicion_Etiqueta_Frecuencia.push(frecuencia);
                }

                // --- Genera valores de etiquetas de Frecuencias ---
                for (i=Math.round(LN_LOG*Math.log(f_min)); i<=Math.round(LN_LOG*

  Math.log(f_max)); i++) {
                        valor_Etiqueta_Frecuencia.push(i);
                }
```

```
                                // --- Colocar etiqueta de frecuencia ---
                                generar_Etiquetas_Frecuencia(sp,posicion_Etiqueta_Frecuencia,
valor_Etiqueta_Frecuencia,
altura_Graficas_MagFase);

                                return sp;
                        }

                        private function generar_Etiquetas_Frecuencia(sp:Sprite,
                posicion:Array,
                valor:Array,
posicion_etiqueta_y:Number):void {

                                // --- Define nuevo "sprite" para escritura de etiquetas ---
                                var sp_Etiqueta_Frecuencia:Sprite = new Sprite();

                                // --- Posicionamiento del Sprite "sp" debajo de la grafica ---
                                sp_Etiqueta_Frecuencia.x = -10;
                                sp_Etiqueta_Frecuencia.y = posicion_etiqueta_y;

                                sp.addChild(sp_Etiqueta_Frecuencia);

                                var campo_Texto:TextField;
                                var formato_Texto:TextFormat = new TextFormat();

                                // --- Valor numerico de la etiqueta ---
                                var valor_Etiqueta_Frecuencia:Number;

                                // --- Seleccion y colocacion de etiquetas ---
                                for (i=0; i<posicion.length; i++) {
                                        campo_Texto = new TextField();
                                        sp_Etiqueta_Frecuencia.addChild(campo_Texto);
                                        // ------------------------------------------------------------
                                        //                                              Seleccion del rango de frecuencia
                                        // ------------------------------------------------------------
                                        if (i>3 && i<7) {
                                                // --- Para 3<i<7: formato = 1KHz hasta 100KHz ---
                                                valor_Etiqueta_Frecuencia = Math.pow(10,valor[i-3]);
                                                if (valor_Etiqueta_Frecuencia >= 1) {
                                                        // --- Formato = 1KHz hasta 100KHz ---
                                                        campo_Texto.text = Math.pow(10,valor[i-3]).toString()+"K";
                                                } else {
                                                        // --- Formato = 1Hz hasta 100Hz ---
                                                        campo_Texto.text = Math.pow(10,valor[i]).toString();
                                                }
                                        } else if (i>=7) {
                                                // --- Para 7<i: formato = 1MHz hasta 100MHz ---
                                                valor_Etiqueta_Frecuencia = Math.pow(10,valor[i-6]);
                                                if (valor_Etiqueta_Frecuencia >= 1) {
                                                        // --- Formato = 1MHz hasta 100MHz ---
                                                        campo_Texto.text = Math.pow(10,valor[i-6]).toString()+"M";
                                                } else {
                                                        // --- Formato = 1KHz hasta 100KHz ---
                                                        campo_Texto.text = Math.pow(10,valor[i-3]).toString()+"K";
                                                }
                                        } else {
                                                // --- Formato = 0.11Hz hasta 0.001Hz ---
                                                campo_Texto.text = Math.pow(10,valor[i]).toString();
                                        }
                                        // ------------------------------------------------------------
                                        //                                              Asignacion de propiedades del texto
                                        // ------------------------------------------------------------
                                        campo_Texto.x = posicion[i];
                                        campo_Texto.y = 0;
```

```
                    asigna_Formato_Texto(campo_Texto,formato_Texto,"CENTER");
                }
            }

            private function dibuja_Titulos_Grafica(Titulo:String,TituloY:String,
TituloX:String,
TituloNodo:String=" "):Sprite {

                var sp:Sprite = new Sprite();

                // --- Define nuevos "sprite" para escritura de Titulos ---
                var sp_Titulo:Sprite = new Sprite();
                var sp_TituloY:Sprite = new Sprite();
                var sp_TituloX:Sprite = new Sprite();
                var sp_TituloNodo:Sprite = new Sprite();

                // --- Posicionamiento de cada Sprite "sp" en la grafica ---
                // label: "Titulo Principal"
                sp_Titulo.x = numero_Muestras/2;
                sp_Titulo.y = -20;
                // label: "TituloY"
                sp_TituloY.x = -45;
                sp_TituloY.y = -20;
                // label: "TituloX"
                sp_TituloX.x = numero_Muestras/2;
                sp_TituloX.y = altura_Graficas_MagFase + 15;
                // label: "TituloNodo"
                sp_TituloNodo.x = 300          //numero_Muestras/3;
                sp_TituloNodo.y = -20;         //altura_Graficas_MagFase + 40;

                sp.addChild(sp_Titulo);
                sp.addChild(sp_TituloY);
                sp.addChild(sp_TituloX);
                sp.addChild(sp_TituloNodo);

                var campo_Texto:TextField;
                var formato_Texto:TextFormat = new TextFormat();

                // --- Titulo de la Grafica ---
                campo_Texto = new TextField();
                sp_Titulo.addChild(campo_Texto);
                campo_Texto.text = Titulo;
                asigna_Formato_Texto(campo_Texto,formato_Texto,"CENTER");

                // --- Titulo de las acotaciones en el eje Y ---
                campo_Texto = new TextField();
                sp_TituloY.addChild(campo_Texto);
                campo_Texto.text = TituloY;
                asigna_Formato_Texto(campo_Texto,formato_Texto,"CENTER");

                // --- Titulo de las acotaciones en el eje X ---
                campo_Texto = new TextField();
                sp_TituloX.addChild(campo_Texto);
                campo_Texto.text = TituloX;
                asigna_Formato_Texto(campo_Texto,formato_Texto,"CENTER");

                // --- Titulo del Nodo cuya grafica se esta presentando---
                campo_Texto = new TextField();
                sp_TituloNodo.addChild(campo_Texto);
                campo_Texto.text = TituloNodo;
                asigna_Formato_Texto(campo_Texto,formato_Texto,"CENTER");

                sp.name = "titulo";

                return sp;
            }

            private function asigna_Formato_Texto(campo_Texto:TextField,
```

```
formato_Texto:TextFormat,

alineacion:String):void {
                                formato_Texto.font = "_sans";
                                formato_Texto.size = 10
                                campo_Texto.width = 25;
                                campo_Texto.textColor = 0xffffff;
                                switch (alineacion) {
                                        case "CENTER":
                                                campo_Texto.autoSize = TextFieldAutoSize.CENTER;
                                        break;
                                        case "LEFT":
                                                campo_Texto.autoSize = TextFieldAutoSize.LEFT;
                                        break;
                                        case "RIGHT":
                                                campo_Texto.autoSize = TextFieldAutoSize.RIGHT;
                                        break;
                                }
                                campo_Texto.setTextFormat(formato_Texto);
                        }

                        private function generar_Etiquetas_Y(sp:Sprite,posicion_Etiqueta:Array,

valor_Etiqueta:Array):void {

                                // --- Define nuevo "sprite" para escritura de etiquetas ---
                                var sp_Etiqueta:Sprite = new Sprite();
                                sp_Etiqueta.x = -25;
                                sp_Etiqueta.y = -8;
                                sp.addChild(sp_Etiqueta);

                                var campo_Texto:TextField;
                                var formato_Texto:TextFormat = new TextFormat();

                                // --- Imprime etiquetas ---
                                for (i=0; i<posicion_Etiqueta.length; i++) {
                                        campo_Texto= new TextField();
                                        sp_Etiqueta.addChild(campo_Texto);
                                        campo_Texto.text = valor_Etiqueta[i].toString();
                                        campo_Texto.y = posicion_Etiqueta[i];
                                        asigna_Formato_Texto(campo_Texto,formato_Texto,"RIGHT");
                                }
                        }

                        // ------------------------------------------------------------
                        //
                        //              Seccion de graficacion de la funcion de Respuesta en
                        //                                              Frecuencia (magnitud o fase)
                        //
                        // ------------------------------------------------------------

                        private function draw_Funcion(arg:Array):Sprite {

                                // ------------------------------------------------------------
                                //  Escalamiento de la Funcion de Respuesta en Frecuencia
                                //  (magnitud y fase)
                                // ------------------------------------------------------------
                                // --- Recepcion de datos ---
                                var min_Acotamiento:Number = arg[0];
                                var max_Acotamiento:Number = arg[1];
                                var delt:Number = arg[2];
                                var max_min:Array = arg[3];
                                var nodo:int = arg[4];
                                var tipo:int = arg[5] - 3;

                                // --- Variables para graficacion de la funcion ---
                                var sp:Sprite = new Sprite();
                                var acotamiento:Sprite = new Sprite();
                                var funcion:Sprite = new Sprite();
```

```
sp.addChild(acotamiento);

// --- Auxiliares ---
var delta_Acotamiento:Number;
var separacion_Lineas:Number;

// --- Valores maximo y minimo para acotar en el eje Y ---
var valor_max_Acotamiento:Number;
var valor_min_Acotamiento:Number;

// --- Calcula posiciones de etiquetas ---
var posicion_Etiqueta:Array = new Array();
var valor_Etiqueta:Array = new Array();

// --- Selecciona acotamiento proximo superior al valor maximo ---
if (max_min[0] == -Infinity) {
	max_min[0] = max_Acotamiento;
	max_min[1] = min_Acotamiento;
}

for (i=min_Acotamiento; i<=max_Acotamiento; i+=delt) {
	if((i-max_min[0])>=0) {
		valor_max_Acotamiento = i;
		break;
	}
}

// --- Selecciona acotamiento proximo inferior al valor minimo ---
for (i=max_Acotamiento; i>=min_Acotamiento; i-=delt) {
	if((i-max_min[1])<=0) {
		valor_min_Acotamiento = i;
		break;
	}
}

// --- Comprueba si valor_max_Acotamiento = valor_min_Acotamiento ---
if (valor_max_Acotamiento == valor_min_Acotamiento) {
	valor_max_Acotamiento = valor_min_Acotamiento + delt;
	valor_min_Acotamiento = valor_min_Acotamiento - delt;
}

delta_Acotamiento = (valor_max_Acotamiento-valor_min_Acotamiento)/delt;

// --- Traza lineas horizontales y posiciona etiquetas ---
acotamiento.graphics.lineStyle(1,color_Lineas_Acotaciones,alfa);
for (k=0; k<delta_Acotamiento; k++) {
	separacion_Lineas = (altura_Graficas_MagFase/delta_Acotamiento)*k;
	acotamiento.graphics.moveTo(0,separacion_Lineas);
	acotamiento.graphics.lineTo(ancho_Graficas_MagFase,
separacion_Lineas);
	posicion_Etiqueta.push(separacion_Lineas);
}
posicion_Etiqueta.push(altura_Graficas_MagFase-5);

// --- Genera valores de etiquetas ---
for (i=valor_max_Acotamiento; i>=valor_min_Acotamiento; i-=delt) {
	valor_Etiqueta.push(i);
}

// --- Imprime etiqueta en las posiciones calculadas ---
generar_Etiquetas_Y(acotamiento,posicion_Etiqueta,valor_Etiqueta);

// --- Grafica de la Funcion ---
funcion = graficar_Funcion(funcion_a_Graficar[nodo][tipo],
                           valor_min_Acotamiento,
                           valor_max_Acotamiento);
sp.addChild(funcion);
```

```
                return sp;
        }

        private function calcula_Maximo_Minimo(datos:Array):Array {

                // --- Auxiliares ---
                var aux:Array = new Array();
                var maximo:Number;
                var minimo:Number;

                aux = datos.concat();

                aux.sort(Array.DESCENDING | Array.NUMERIC);
                maximo = Math.round(aux[0]*100)/100;
                minimo = Math.round(aux[aux.length-1]*100)/100;

                return new Array(maximo,minimo);
        }

        private function graficar_Funcion(datos:Array,

valor_min_Acotamiento:Number,

valor_max_Acotamiento:Number):Sprite {

                var sp:Sprite = new Sprite();

                var fx:Number;        // Funcion a graficar
                var pendiente:Number;
                var ordenada:Number;
                var top_margin:Number = 10;

                datos_Graficacion = new Array();

                // ------------------------------------------------------------
                //   Funcion(frecuencia) = pendiente*frecuencia + ordenada
                //                normaliza mediante los valores maximo y minimo de
                //                                                acotamiento de la magnitud
                // ------------------------------------------------------------
                pendiente = altura_Graficas_MagFase/(valor_max_Acotamiento -

valor_min_Acotamiento);
                ordenada = -pendiente*valor_max_Acotamiento +

altura_Graficas_MagFase;
                // ------------------------------------------------------------
                //  Grafica de la Respuesta en Frecuencia de la variable
                // ------------------------------------------------------------
                sp.graphics.lineStyle(1,color_Grafica);
                fx = pendiente*datos[0] + ordenada;
                sp.graphics.moveTo(0,altura_Graficas_MagFase-fx);
                for (i=1; i<LD.LF.length; i++) {
                        fx = pendiente*datos[i] + ordenada;
                        sp.graphics.lineTo(i,altura_Graficas_MagFase-fx);
                        datos_Graficacion.push(altura_Graficas_MagFase-fx);
                }
                return sp;
        }

        // ------------------------------------------------------------
        //
        // Conexion entre "graficaBarras" --> "Generador_Respuesta_Frecuencia"
        // (activacion del "texto" que identifica a la grafica de Respuesta
        //  en Frecuencia correspondiente al nodo seleccionado en la
        // "graficaBarras")
        //
        // ------------------------------------------------------------

        public function mostrarNodoSeleccionado(id:String):void {
                if (!flagInicio) {
```

```
                removeChild(getChildByName("nodoSeleccionadoMagnitud"));
                removeChild(getChildByName("nodoSeleccionadoFase"));
            }

            // ----------------------------------------------------------
            // Para cada nodo seleccionado en la "graficaBarras", se
            // muestra el "texto" encerraddo en un cuadro blanco con fondo
            // azul, correspondiente a las graficas de la magntiud y fase
            // de la respuesta en frecuencia de dicho nodo
            // ----------------------------------------------------------
            // --- Nodo Seleccionado Magnitud ---
            var nodoSeleccionadoMagnitud:TextField =new TextField();
            nodoSeleccionadoMagnitud.text = "V"+id.substr(1,2);
            nodoSeleccionadoMagnitud.x = g_Magnitud.x + 15;
            nodoSeleccionadoMagnitud.y = g_Magnitud.y -20;
            nodoSeleccionadoMagnitud.autoSize = TextFieldAutoSize.CENTER;
            nodoSeleccionadoMagnitud.border = true;
            nodoSeleccionadoMagnitud.borderColor = 0xffffff;
            nodoSeleccionadoMagnitud.background = true;
            nodoSeleccionadoMagnitud.backgroundColor = 0x0099cc;
            nodoSeleccionadoMagnitud.width = 20;
            nodoSeleccionadoMagnitud.height = 15;
            nodoSeleccionadoMagnitud.name = "nodoSeleccionadoMagnitud";

            // --- Formato de Texto Magnitud ---
            var formatMagnitud:TextFormat = new TextFormat();
            formatMagnitud.font = "_sans";
            formatMagnitud.size = 10;
            formatMagnitud.color = 0xffffff;
            nodoSeleccionadoMagnitud.setTextFormat(formatMagnitud);
            addChild(nodoSeleccionadoMagnitud);

            // --- Nodo Seleccionado Fase ---
            var nodoSeleccionadoFase:TextField =new TextField();
            nodoSeleccionadoFase.text = "V"+id.substr(1,2);
            nodoSeleccionadoFase.x = g_Fase.x + 15;
            nodoSeleccionadoFase.y = g_Fase.y - 20;
            nodoSeleccionadoFase.autoSize = TextFieldAutoSize.CENTER;
            nodoSeleccionadoFase.border = true;
            nodoSeleccionadoFase.borderColor = 0xffffff;
            nodoSeleccionadoFase.background = true;
            nodoSeleccionadoFase.backgroundColor = 0x0099cc;
            nodoSeleccionadoFase.width = 20;
            nodoSeleccionadoFase.height = 15;
            nodoSeleccionadoFase.name = "nodoSeleccionadoFase";

            // --- Formato de Texto Fase ---
            var formatFase:TextFormat = new TextFormat();
            formatFase.font = "_sans";
            formatFase.size = 10;
            formatFase.color = 0xffffff;
            nodoSeleccionadoFase.setTextFormat(formatFase);
            addChild(nodoSeleccionadoFase);

            flagInicio = false;
        }

    }
}
```

```
package Vistas.ac.respuestaFrecuencia
{
	// ************************************************************
	//
	//		Clase:	Cursor_Respuesta_Frecuencia
	//
	//		Patron de Programacion:
	//
	//			Auxiliar de la clase "Graficador_Respuesta_Frecuencia"
	//
	//
	abril / 2010
	// ************************************************************
	//
	//		ENTRADA:
	//		  - Referencia a la grafica de la clase
	//			"Graficador_Respuesta_Frecuencia"
	//
	//		- LF --> Array: Valores de Frecuencia en variacion logaritmica
	//					(de "fmin" a "fmax")
	//		* LD[i][3][indiceFrecuencia] = Magnitud_dB:		Number
	//		* LD[i][4][indiceFrecuencia] = Fase_Grados:	Number
	//
	//		CONEXIONES INTERACTIVAS:
	//					- Con la clase "Graficador_Respuesta_Frecuencia"
	//
	//		SALIDA:
	//			- Referencia a la clase "CURSOR" (cursor interactivo)
	//			  * Cursor activo sobre las graficas de magnitud y fase
	//			  * Texto informativo: (magnitud/fase vs. frecuencia)
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Campos de texto para etiquetas
	import flash.text.*;
	// Eventos
	import flash.events.*;

	// Modelo y Vistas
	import Modelo.Generador_Lista_Datos;
	import Vistas.ac.vistasAC;

	 public class Cursor_Respuesta_Frecuencia extends Sprite {

		// --- Graficas de Magnitud y Fase ---
		public var g_Magnitud:Sprite;
		public var g_Fase:Sprite;

		// --- Cursores ---
		private var Magnitud:CURSOR;
		private var Fase:CURSOR;
		private var cursor_Magnitud:Sprite;
		private var cursor_Fase:Sprite;
		// --- Grupo de cursores encadenados ---
```

```
		private var grupo_Cursores:Array;

		// --- Valores de Magnitud y Fase ---
		private var LD:Generador_Lista_Datos;
		private var datos_Frecuencia:Array;
		private var datos_Magnitud:Array;
		private var datos_Fase:Array;
		private var graficas_Frecuencia:Array;

		// grafica.funcion_a_Graficar[nodo][magnitud] --> Datos Magnitud_dB
		private var magnitud:int = 0;
		// grafica.funcion_a_Graficar[nodo][fase] --> Datos Fase_Grados
		private var fase:int = 1;
		private var nodo:Number;

		// --- Indices ---
		private var i:int;
		private var j:int;
		private var k:int;

		// --- Parametros de graficas ---
		public var altura_Graficas_MagFase:Number;
		public var ancho_Graficas_MagFase:Number;

		// --- Colores ---
		// --- Fondo ----------------------
		public var color_Fondo:Number;
		public var alfa_Fondo:Number;
		public var color_Linea:Number;
		public var alfa_Linea:Number;
		// --- Punto de interseccion -------
		public var color_Fondo_Punto:Number;
		public var alfa_Fondo_Punto:Number;
		public var color_Linea_Punto:Number;
		public var alfa_Linea_Punto:Number;

		// Referencia para envio de actualizacion de "coordenadaX"
		// del CURSOR
		public var vistas_ac:vistasAC;

		// --- Composicion: Referencia a "Graficador_Respuesta_Frecuencia" ---
		private var grafica:Graficador_Respuesta_Frecuencia;

		public function Cursor_Respuesta_Frecuencia(_vistas_ac:vistasAC,

_LD:Generador_Lista_Datos,
                                                    _grafica:Graficador_Respuesta_Frecuencia):void {

			vistas_ac = _vistas_ac;

			// --- Recepcion de datos ---
			grafica = _grafica;
			graficas_Frecuencia = grafica.graficas_Frecuencia;

			// --- Acceso a parametros de graficas ---
			altura_Graficas_MagFase = grafica.altura_Graficas_MagFase;
```

```
                        ancho_Graficas_MagFase = grafica.ancho_Graficas_MagFase;

                        // --- Acceso a composicion de graficas ---
                        g_Magnitud = grafica.g_Magnitud;
                        g_Fase = grafica.g_Fase;

                        // --- Acceso a la Lista de Datos ---
                        // --- Lista de Datos ---
                        LD = _LD;
                        // grafica.funcion_a_Graficar[nodo][magnitud] --> Datos Magnitud_dB
                        magnitud = 0;
                        // grafica.funcion_a_Graficar[nodo][fase] --> Datos Fase_Grados
                        fase = 1;
                        // Datos de frecuencia
                        datos_Frecuencia = LD.LF;

                        grupo_Cursores = new Array();

                        // ------------------------------------------------------------
                        //                              Construccion del cursor de Magnitud
                        // ------------------------------------------------------------
                        Magnitud = new CURSOR(this,"Magnitud",altura_Graficas_MagFase,

ancho_Graficas_MagFase,datos_Frecuencia);
                        Magnitud.construye_Cursor();
                        Magnitud.construye_Texto("Mag = "," dB","f = "," Hz");
                        cursor_Magnitud = new Sprite();
                        grupo_Cursores.push(Magnitud);

                        // ------------------------------------------------------------
                        //                              Construccion del cursor de Fase
                        // ------------------------------------------------------------
                        Fase = new CURSOR(this,"Fase",altura_Graficas_MagFase,

ancho_Graficas_MagFase,datos_Frecuencia);
                        Fase.construye_Cursor();
                        Fase.construye_Texto("Fase = "," Gr","f = "," Hz");
                        cursor_Fase = new Sprite();
                        grupo_Cursores.push(Fase);

                        // ------------------------------------------------------------
                        //                              Encadenamiento de cursores
                        // ------------------------------------------------------------
                        Magnitud.encadenar_Cursores(grupo_Cursores);
                        Fase.encadenar_Cursores(grupo_Cursores);

                        // --- Colores por default ---
                        color_Fondo = 0x0099ff;
                        alfa_Fondo = 0.4;
                        color_Linea = 0xffffff;
                        alfa_Linea = 1;
                        color_Fondo_Punto = 0xcccccc;
                        alfa_Fondo_Punto = 0.4;
                        color_Linea_Punto = 0xffffff;
                        alfa_Linea_Punto = 1;
                }
```

```
// ----------------------------------------------------------
//
//                          Seccion de Cursores interactivos
//
// ----------------------------------------------------------

public function inicializaCursorInteractivo():void {

        var referencias_Cursores:Array = new Array();

        // ----------------------------------------------------------
        //                          Formacion del cursor de Magnitud
        // ----------------------------------------------------------
        cursor_Magnitud.addChild(Magnitud.cursor);
        cursor_Magnitud.addChild(Magnitud.texto);

        // ----------------------------------------------------------
        //                          Formacion del cursor de Fase
        // ----------------------------------------------------------
        cursor_Fase.addChild(Fase.cursor);
        cursor_Fase.addChild(Fase.texto);

        // ----------------------------------------------------------
        // Despliega la grafica del nodo de tierra del circuito,
        // por defaul
        // ----------------------------------------------------------
        //                                  Grafica de Magnitud
        // ----------------------------------------------------------
        // --- Layer 4: Funcion y Acotamiento de Magnitud
        nodo = graficas_Frecuencia.length-1;
        g_Magnitud.addChild(graficas_Frecuencia[nodo].Magnitud_dB);
        g_Magnitud.addChild(cursor_Magnitud);

    Magnitud.actualiza_Datos(graficas_Frecuencia[nodo].datos_Grafica_Magnitud,
grafica.funcion_a_Graficar[nodo][magnitud]);

        // ----------------------------------------------------------
        //                                  Grafica de Fase
        // ----------------------------------------------------------
        // --- Layer 4: Funcion y Acotamiento de Fase
        g_Fase.addChild(graficas_Frecuencia[nodo].Fase_Grados);
        g_Fase.addChild(cursor_Fase);
        Fase.actualiza_Datos(graficas_Frecuencia[nodo].datos_Grafica_Fase,
grafica.funcion_a_Graficar[nodo][fase]);

        // ----------------------------------------------------------
        //                                  Animacion inical de cursores
        // ----------------------------------------------------------
        Magnitud.animacion_inicial();

        // ----------------------------------------------------------
        //                                  Interactividad de cursores
        // ----------------------------------------------------------
```

```
                        Magnitud.cursor.addEventListener(MouseEvent.MOUSE_DOWN,
Magnitud.mover_cursor);
                        Magnitud.cursor.addEventListener(MouseEvent.MOUSE_UP,
Magnitud.detener_cursor);
                        Magnitud.cursor.addEventListener(MouseEvent.ROLL_OUT,
Magnitud.detener_cursor);

                        Fase.cursor.addEventListener(MouseEvent.MOUSE_DOWN,
Fase.mover_cursor);
                        Fase.cursor.addEventListener(MouseEvent.MOUSE_UP,
Fase.detener_cursor);
                        Fase.cursor.addEventListener(MouseEvent.ROLL_OUT,
Fase.detener_cursor);
                }

                // ----------------------------------------------------------
                //
                //          Conexion entre:
                //            "graficaBarras" --> "Generador_Respuesta_Frecuencia"
                //           (activacion de la grafica de Respuesta en Frecuencia
                // correspondiente al nodo seleccionado en la "graficaBarras")
                //
                // ----------------------------------------------------------

                public function selecciona_Grafica_Nodo(id:String):void {

                        // ----------------------------------------------------------
                        // Para cada nodo seleccionado en la "graficaBarras", se
                        // muestra la grafica correspondiente de la magntiud y fase
                        // de la respuesta en frecuencia de dicho nodo
                        // ----------------------------------------------------------
                        nodo = Number(id.substr(1,2));

                        // --- Remueve la grafica anterior -------------------------
                        g_Magnitud.removeChildAt(g_Magnitud.numChildren-1); // Cursor
                        g_Magnitud.removeChildAt(g_Magnitud.numChildren-1);          // Grafica
                        g_Fase.removeChildAt(g_Fase.numChildren-1);
                        g_Fase.removeChildAt(g_Fase.numChildren-1);
                        // --- Agrega la grafica del nodo seleccionado -------------
                        g_Magnitud.addChild(graficas_Frecuencia[nodo].Magnitud_dB);
                        g_Magnitud.addChild(cursor_Magnitud);
                        g_Fase.addChild(graficas_Frecuencia[nodo].Fase_Grados);
                        g_Fase.addChild(cursor_Fase);

                        // --- Actualizar datos a desplegar ------------------------

        Magnitud.actualiza_Datos(graficas_Frecuencia[nodo].datos_Grafica_Magnitud,
grafica.funcion_a_Graficar[nodo][magnitud]);
                        Fase.actualiza_Datos(graficas_Frecuencia[nodo].datos_Grafica_Fase,
```

```
grafica.funcion_a_Graficar[nodo][fase]);
                }

                private function actualiza_Grafica_Nodo(event:MouseEvent):void {

                        nodo = Number(DisplayObject(event.target).name);

                        // --- Remueve la grafica anterior ------------------------
                        g_Magnitud.removeChildAt(g_Magnitud.numChildren-1); // Cursor
                        g_Magnitud.removeChildAt(g_Magnitud.numChildren-1);          // Grafica
                        g_Fase.removeChildAt(g_Fase.numChildren-1);
                        g_Fase.removeChildAt(g_Fase.numChildren-1);
                        // --- Agrega la grafica del nodo seleccionado -------------
                        g_Magnitud.addChild(graficas_Frecuencia[nodo].Magnitud_dB);
                        g_Magnitud.addChild(cursor_Magnitud);
                        g_Fase.addChild(graficas_Frecuencia[nodo].Fase_Grados);
                        g_Fase.addChild(cursor_Fase);

                        // --- Actualizar datos a desplegar ------------------------

        Magnitud.actualiza_Datos(graficas_Frecuencia[nodo].datos_Grafica_Magnitud,

grafica.funcion_a_Graficar[nodo][magnitud]);
                        Fase.actualiza_Datos(graficas_Frecuencia[nodo].datos_Grafica_Fase,

grafica.funcion_a_Graficar[nodo][fase]);
                }

public function actualiza_CoordX_CURSOR(coordenadaX:Number):void {
        vistas_ac.actualiza_CoordX_CURSOR(coordenadaX);
}

         }
}
```

```
package Vistas.ac.respuestaFrecuencia
{
	// ************************************************************
	//
	//		Clase:	CURSOR
	//
	//		Patron de Programacion:
	//
	//			Auxiliar de la clase "Cursor_Respuesta_Frecuencia"
	//
	//
	abril / 2010
	// ************************************************************
	//
	//		ENTRADA:
	//		  - Referencia a la clase "Cursor_Respuesta_Frecuencia"
	//			* Valores de Frecuencia en variacion logaritmica
	//			* Magnitud_dB
	//			* Fase_Grados
	//
	//		CONEXIONES INTERACTIVAS:
	//						- Con la clase "Cursor_Respuesta_Frecuencia"
	//
	//		SALIDA:			- Cursor activo sobre las graficas de magnitud y fase
	//						- Texto informativo: (magnitud/fase vs. frecuencia)
	//
	// ************************************************************

	// Flash principal
	import flash.display.*;
	// Campos de texto para etiquetas
	import flash.text.*;
	// Eventos
	import flash.events.*;
	// Timer
	import flash.utils.*;

	 public class CURSOR extends Sprite {

		// --- Cursores ---
		// Layer 1:
		public var cursor:Sprite;
		public var fondo:Sprite;
		public var marcador:Sprite;
		// Layer 2:
		public var texto:Sprite;

		// --- Conexion de cursores ---
		private var grupo_Cursores:Array;

		// --- Animacion ---
		private var tm0:Timer;

		// --- Lista de Datos ---
		private var datosX:Array;		// Eje "x" de la grafica
		private var datosY:Array;		// Eje "y" de la grafica
```

```
		public var coordenadaX:Number;

		// --- Posicion Marcador ---
		private var datosY_marcador:Array;	// Posicion del "marcador"
                                                                                // sobre la
grafica actual

		// --- Indices ---
		private var i:int;
		private var j:int;
		private var k:int;

		public var altura_Grafica:Number;
		public var ancho_Grafica:Number;

		// --- Colores ---
		// --- Fondo ---
		public var color_Fondo:Number;
		public var alfa_Fondo:Number;
		public var color_Linea:Number;
		public var alfa_Linea:Number;
		// --- Punto de interseccion ---
		public var color_Fondo_Punto:Number;
		public var alfa_Fondo_Punto:Number;
		public var color_Linea_Punto:Number;
		public var alfa_Linea_Punto:Number;

		// --- Auxiliares ---
		private var campo_Texto1:TextField;
		private var campo_Texto2:TextField;
		private var formato_Texto:TextFormat;
		private var frecuencia:Number;

		private var ancho_Caja_Texto:Number = 90;
		private var alto_Caja_Texto:Number = 30;

		// --- Leyendas de texto ---
		private var leyenda1:String;
		private var unidades_leyenda1:String;
		private var leyenda2:String;
		private var unidades_leyenda2:String;

		// Referencia para envio de actualizacion de
		// "coordenadaX" del CURSOR
		public var cursorRespuestaFrecuencia:Cursor_Respuesta_Frecuencia;

		public function
CURSOR(_cursorRespuestaFrecuencia:Cursor_Respuesta_Frecuencia,
                                                        _id:String,altura:Number,
                                                                ancho:Number,_datosX:Array):void {

			// --- Recepcion de datos ---
			cursorRespuestaFrecuencia = _cursorRespuestaFrecuencia;

			altura_Grafica = altura;
			ancho_Grafica = ancho;
```

```
        datosX = _datosX;

        // --- Inicializacion de cursores ---
        // Layer 1:
        cursor = new Sprite();
        cursor.name = _id;
        fondo = new Sprite();
        marcador = new Sprite();

        // Layer 2:
        texto = new Sprite();
        campo_Texto1 = new TextField();
        campo_Texto2 = new TextField();
        formato_Texto = new TextFormat();

        // --- Colores por default ---
        color_Fondo = 0x0099ff;
        alfa_Fondo = 0.4;
        color_Linea = 0xffffff;
        alfa_Linea = 1;
        color_Fondo_Punto = 0xcccccc;
        alfa_Fondo_Punto = 0.4;
        color_Linea_Punto = 0xffffff;
        alfa_Linea_Punto = 1;

        // -- Iniciliza "timer" para animacion ---
        tm0 = new Timer(1000,1);

}

// ----------------------------------------------------------
//
//          Seccion de construccion de "cursores interactivos"
//                                    y "textos informativos de cursores"
//
// ----------------------------------------------------------

public function construye_Cursor():Sprite {

        // ----------------------------------------------------------
        //                                    Construccion del cursor azul
        // ----------------------------------------------------------
        // --- Layer 1: --> sublayer: "Fondo" -------------------------
        fondo.graphics.lineStyle(1,color_Linea,0.4);
        fondo.graphics.beginFill(color_Fondo,alfa_Fondo)
        fondo.graphics.drawRect(-10,0,20,-altura_Grafica);
        fondo.graphics.endFill();
        fondo.y = altura_Grafica;
        // --- Linea central ---
        fondo.graphics.lineStyle(1,color_Linea,alfa_Linea);
        fondo.graphics.moveTo(0,0);
        fondo.graphics.lineTo(0,-altura_Grafica);

        // ----------------------------------------------------------
        //                                    Construccion del "Punto mobil"
        // ----------------------------------------------------------
```

```
                // --- Layer 1: --> sublayer: "Punto" -------------------------
                // --- Punto de interseccion con la grafica ---
                marcador.graphics.lineStyle(1,color_Linea_Punto,0.5);
                marcador.graphics.beginFill(color_Fondo_Punto,alfa_Fondo_Punto)
                marcador.graphics.drawCircle(0,0,5);
                marcador.graphics.endFill();

                // --- Formacion de Layer 1:
                cursor.addChild(fondo);
                cursor.addChild(marcador);

                // --- Interactividad ---
                cursor.buttonMode = true;
                cursor.visible = false;

                return cursor;
        }

        public function construye_Texto(_leyenda1:String,

_unidades_leyenda1:String,
                                                                _leyenda2:String,

_unidades_leyenda2:String):Sprite {

                // ------------------------------------------------------------
                //                          Construccion del texto informativo
                // ------------------------------------------------------------
                leyenda1 = _leyenda1;
                unidades_leyenda1 = _unidades_leyenda1;
                leyenda2 = _leyenda2;
                unidades_leyenda2 = _unidades_leyenda2;

                // --- Layer 2: ---
                // --- Marco de texto ---
                texto.graphics.lineStyle(2,0xffffff,0.8);
                texto.graphics.beginFill(color_Fondo,alfa_Fondo);
                texto.graphics.drawRect(0,0,ancho_Caja_Texto,alto_Caja_Texto);
                texto.graphics.endFill();

                // --- Formacion de Layer 2 ---
                texto.addChild(campo_Texto1);
                texto.addChild(campo_Texto2);
                texto.alpha = 0;

                // --- Define formato para etiquetas ---
                formato_Texto.font = "_sans";
                formato_Texto.size = 10;
                formato_Texto.color = 0xffffff;

                // --- campo_Texto1 ---
                campo_Texto1.x = 5;
                campo_Texto1.y = 0;
                campo_Texto1.height = 12;
                campo_Texto1.width = 20;
                campo_Texto1.autoSize = TextFieldAutoSize.LEFT;
```

```
                campo_Texto1.setTextFormat(formato_Texto);

                // --- campo_Texto2 ---
                campo_Texto2.x = 5;
                campo_Texto2.y = 12;
                campo_Texto2.height = 12;
                campo_Texto2.width = 20;
                campo_Texto2.autoSize = TextFieldAutoSize.LEFT;
                campo_Texto2.setTextFormat(formato_Texto);

                return texto;
        }

        // ----------------------------------------------------------
        //
        //                      Seccion de actualizacion de datos de
        // "cursores interactivos" y "textos informativos de cursores"
        //
        // ----------------------------------------------------------

        public function actualiza_Datos(_datosY_marcador:Array,
                                                                        _datosY:Array):void
{
                // ----------------------------------------------------------
                //              Actualizacion de datos de "magnitud" o "fase"
                // ----------------------------------------------------------
                datosY_marcador = _datosY_marcador;
                datosY = _datosY;
                actualizar();
        }

        private function actualiza_Texto(mY:Number,mX:Number): void {
                // ----------------------------------------------------------
                //              Actualizacion de textos de "magnitud" o "fase"
                // ----------------------------------------------------------
                campo_Texto1.text = leyenda1+mY.toString()+unidades_leyenda1;
                campo_Texto1.setTextFormat(formato_Texto);

                campo_Texto2.text = leyenda2+mX.toString()+unidades_leyenda2;
                campo_Texto2.setTextFormat(formato_Texto);
        }

        // ----------------------------------------------------------
        //
        //        Seccion de moviemiento de "cursores interactivos"
        //
        // ----------------------------------------------------------

        public function mover_cursor(event:MouseEvent):void {
                cursor.startDrag();
                cursor.addEventListener(Event.ENTER_FRAME, desplazarX);
        }

        public function detener_cursor(event:MouseEvent):void {
                cursor.stopDrag();
                cursor.y = 0;
```

```
                    cursor.removeEventListener(Event.ENTER_FRAME, desplazarX);

// Comunica "coordenadaX" a vista en 3D
cursorRespuestaFrecuencia.actualiza_CoordX_CURSOR(coordenadaX);

                }

                public function desplazarX(e:Event):void {

                    cursor.y = 0;

                    if (cursor.x < 0) {
                        cursor.x = 0;
                        cursor.stopDrag();
                    }
                    if (cursor.x > ancho_Grafica-1) {
                        cursor.x = ancho_Grafica-1;
                        cursor.stopDrag();
                        cursor.removeEventListener(Event.ENTER_FRAME, desplazarX);
                    } else {
                        actualizar();
                        conectar_Cursores();
                    }
                }

                private function actualizar():void {
                    // ------------------------------------------------------------
                    //                  Actualizacion de datos de "magnitud" o "fase"
                    // ------------------------------------------------------------
                    coordenadaX = Math.round(cursor.x);

                    marcador.y = datosY_marcador[coordenadaX];

                    // --- Actualiza despliegue de datos -------------------
                    if (datosX[coordenadaX] > 1) {
                        // --- Frecuencia > 1 Hz ---
                        frecuencia = Math.round(datosX[coordenadaX]*100)/100;
                    } else {
                        // --- Frecuencia < 1 Hz ---
                        frecuencia = Math.round(datosX[coordenadaX]*10000)/10000;
                    }
                    actualiza_Texto(datosY[coordenadaX],frecuencia);
                    posiciona_Cajas_Texto();
                }

                public function posiciona_Cajas_Texto():void {
                    // ------------------------------------------------------------
                    //          Posiciona "cajas de texto" en forma dinamica dentro del
                    //                  area de graficacion de la "magnitud" o "fase"
                    // ------------------------------------------------------------
                    if (cursor.x >= 0) {
                        if (cursor.x < ancho_Grafica - ancho_Caja_Texto - 30) {
                            texto.x = cursor.x + 25;
                            if (marcador.y > alto_Caja_Texto + 15) {
                                texto.y = marcador.y - alto_Caja_Texto - 10;
                            } else {
```

```
                                        texto.y = marcador.y + 10;
                                }
                        } else {
                                texto.x = cursor.x - ancho_Caja_Texto - 25;
                                if (marcador.y > alto_Caja_Texto + 15) {
                                        texto.y = marcador.y - alto_Caja_Texto - 10;
                                } else {
                                        texto.y = marcador.y + 10;
                                }
                        }
                }
        }

        // ----------------------------------------------------------
        //
        //          Seccion de encadenamiento de "cursores interactivos"
        //
        // ----------------------------------------------------------

        public function encadenar_Cursores(_grupo_Cursores:Array):void {

                grupo_Cursores = _grupo_Cursores;
        }

        public function conectar_Cursores():void {

                for (i=0; i<grupo_Cursores.length; i++) {
                        if ( grupo_Cursores[i].cursor.name != cursor.name) {
                                grupo_Cursores[i].cursor.x = Math.round(cursor.x);
                                // ----------------------------------------------------------
                                //   Encadenamiento del cursor "slave" al cursor "target"
                                // ----------------------------------------------------------
                                with(grupo_Cursores[i]) {

                                        coordenadaX = Math.round(cursor.x);

                                        cursor.y = 0;

                                        if (cursor.x < 0) {
                                                cursor.x = 0;
                                                cursor.stopDrag();
                                        }
                                        if (cursor.x > ancho_Grafica-1) {
                                                cursor.x = ancho_Grafica-1;
                                                cursor.stopDrag();
                                        }

                                        marcador.y = datosY_marcador[coordenadaX];

                                        // --- Actualiza despliegue de datos ---
                                        if (datosX[cursor.x] > 1) {
                                                frecuencia =
Math.round(datosX[coordenadaX]*100)/100;
                                        } else {
                                                frecuencia =
Math.round(datosX[coordenadaX]*10000)/
```

```
                                                  10000;
                                                }
                                                actualiza_Texto(datosY[coordenadaX],frecuencia);
                                                posiciona_Cajas_Texto();

                                                // --- Incrementa "alpha" de la caja de texto "texto" -
--
                                                if (((ancho_Grafica/2 - cursor.x) <= 1) &&

(texto.alpha < 1)) {
                                                        texto.alpha += 0.1;
                                                }
                                        }
                                }
                        }
                }

                // ----------------------------------------------------------
                //
                //            Seccion de animacion inicial de "cursores interactivos"
                //
                // ----------------------------------------------------------

                public function animacion_inicial():void {

                        // ----------------------------------------------------------------
                        //                                                Animacion inical de cursores
                        // ----------------------------------------------------------------
                        tm0.addEventListener(TimerEvent.TIMER,inicia_retardo);
                        tm0.addEventListener(TimerEvent.TIMER_COMPLETE,termina_retardo);
                        tm0.start();

                }

                private function inicia_retardo(e:TimerEvent):void {
                        cursor.visible = false;
                }

                private function termina_retardo(e:TimerEvent):void {

                        for (i=0; i<grupo_Cursores.length; i++) {
                                grupo_Cursores[i].cursor.visible = true;
                        }
                        // --- Animacion "easing" ---
                        this.addEventListener(Event.ENTER_FRAME,inicia_animacion_1);
                }

                private function inicia_animacion_1(e:Event):void {

                        var dx:Number;
                        var vx:Number;
                        var easing:Number = 0.1;

                        dx = ancho_Grafica/2 + 1 - cursor.x;
```

```
                        if(dx > 1) {
                                vx = dx*easing;
                                cursor.x += vx;
                        } else if (texto.alpha < 1) {
                                texto.alpha += 0.1;
                        } else {
                                stop_animation();
                        }
                        actualizar();
                        conectar_Cursores();
                }

                private function stop_animation():void {
// Comunica "coordenadaX" a vista en 3D
cursorRespuestaFrecuencia.actualiza_CoordX_CURSOR(coordenadaX);
                        this.removeEventListener(Event.ENTER_FRAME,inicia_animacion_1);
                }

        }
}
```