0. PROLOGO

SCICOS es la Toolbox para modelado y simulación del software científico **SCILAB** desarrollado en **INRIA** (Intitute National de Recherche en Informatique et en Automatique),y que es de libre distribución a través de Internet. La forma de obtener e instalar **SCILAB** en el ordenador viene explicada con todo detalle en el trabajo de doctorado **Manual de Introducción al Tratamiento de Señales con Scilab para Usuarios de Matlab**, de este mismo autor.

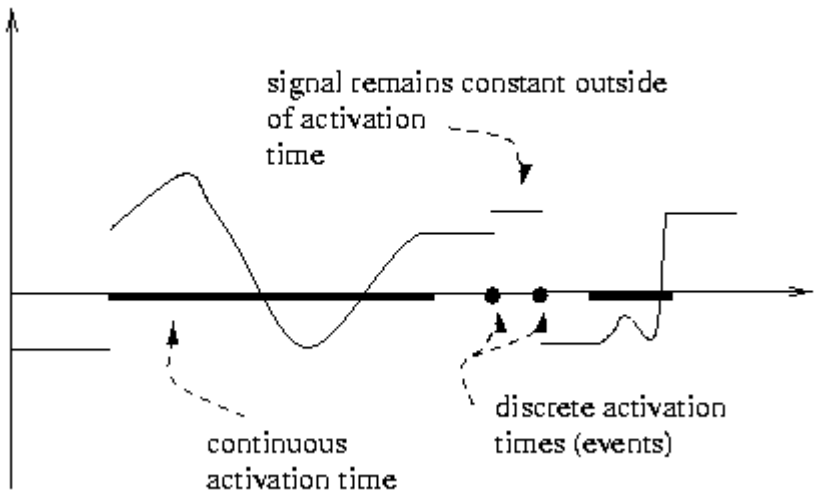
SCICOS modela los sistemas físicos mediante bloques interconectados. El usuario puede crear sus propios bloques que se programan con código **SCILAB**, **C** o **FORTRAN**, siendo también útil para el usuario que domine **MATLAB**, ya que puede desarrollar el código en **MATLAB** y traducirlo a **SCILAB** (REF 1 y 2), mediante una herramienta de traducción que incluye **SCILAB**.

Este trabajo pretende presentar **SCICOS** para que se pueda utilizar como herramienta alternativa a DYMOLA, SIMULINK o ACSL, además de **SCILAB** como hemos comentado anteriormente es una herramienta gratuita y en constante mejora, ya que **INRIA** va actualizando el software con nuevas versiones que contienen nuevas posibilidades.

1. INTRODUCCIÓN.

Scicos (Scilab Connected Object Simulator) es una Toolbox de **Scilab** para el modelado y simulación de sistemas dinámicos incluyendo tanto *sistemas discretos* como *continuos*. **Scicos** incluye un editor gráfico para la construcción de modelos mediante interconexión de **bloques** (***blocks***). Estos bloques representan funciones fundamentales predefinidas en **Scilab** o definidas por el usuario.

En **Scicos** cada señal tiene asociada un conjunto de índices de tiempo, llamados **tiempos de activación** sobre los que la señal puede evolucionar. Fuera de estos *tiempos de activación*, las señales de Scicos permanecen constantes según se muestra en la siguiente figura :



El conjunto de tiempo de activación es una unión de intervalos de tiempo y puntos aislados llamados **eventos** (*events*). Las señales en Scicos son generadas por **bloques** (*blocks*) controlados por señales de activación, que hacen que los **bloques** (*blocks*) produzcan una salida a partir de su entrada y de su estado interno. La señal de salida hereda del bloque que la ha generado, el **conjunto de tiempos de activación** (*activation time set*), pudiéndose usar esta para controlar otros bloques.

A continuación se muestran dos **bloques** (*blocks*) :

Las **entradas para las señales de activación** (*activation input ports*) de los bloques, son los que se encuentran en la parte superior y son de color rojo. Un bloque que no tenga entradas de activación se encuentra permanentemente activo, sin embargo recibe sus tiempos de activación a partir de los de sus **señales de entrada** (*input signals*).

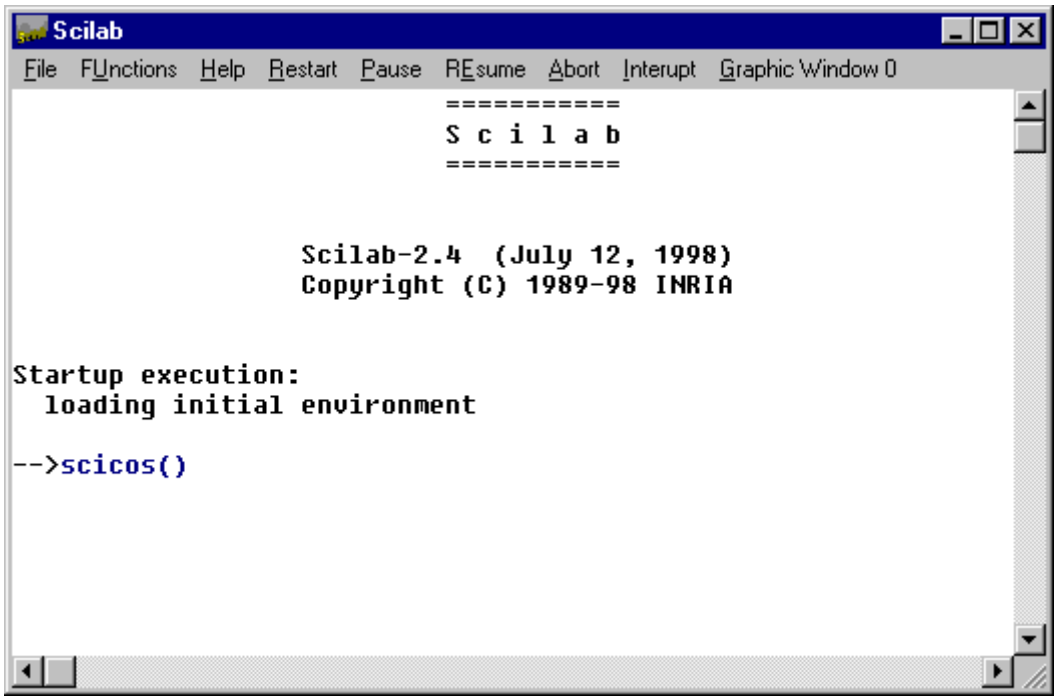
Las **salidas de activación** (*activation output ports*) de un bloque se encuentran en la parte inferior, las señales de salida de estos puertos se consideran por tanto señales de activación generados por los bloques. Por ejemplo, un bloque Reloj (*clock*) genera una señal de activación compuesta de un tren de eventos espaciados regularmente. Si su salida estuviera conectada a la entrada de activación de un bloque de presentación (*Mscope block*), esta señal determinará los tiempos en los que deberá representar gráficamente las señales de entrada.

2. EL EDITOR DE SCICOS.

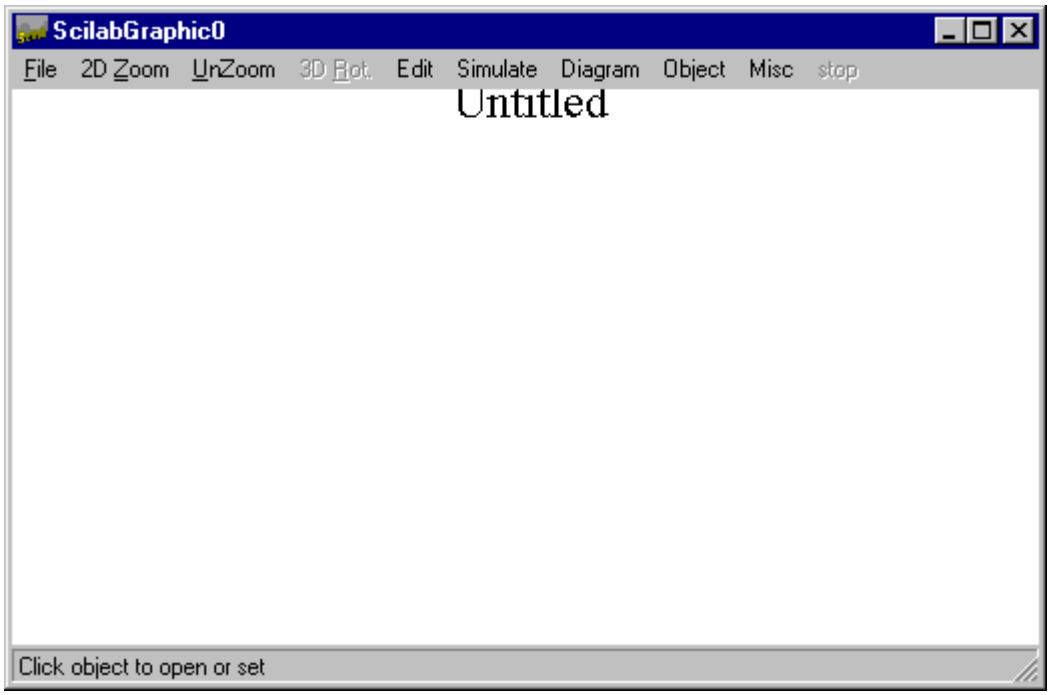
Los sistemas son modelados mediante la interconexión de bloques o subsistemas ; los subsistemas son también llamados **super bloques** (*super blocks*) que forman un único bloque partiendo de un conjunto de bloques simples.

Scicos cuenta con un interfaz gráfico para el usuario fácil de usar, en el que se pueden desarrollar nuevos modelados partiendo de bloques que estén implementados, que se encuentran en las **paletas** (*palettes*) o bien definidos por el usuario.

Para abrir la ventana principal de Scicos debo teclear **scicos()** en Scilab, como muestra la siguiente figura :



Apareciendo **la ventana principal de Scicos** (*Scicos's main window*) :

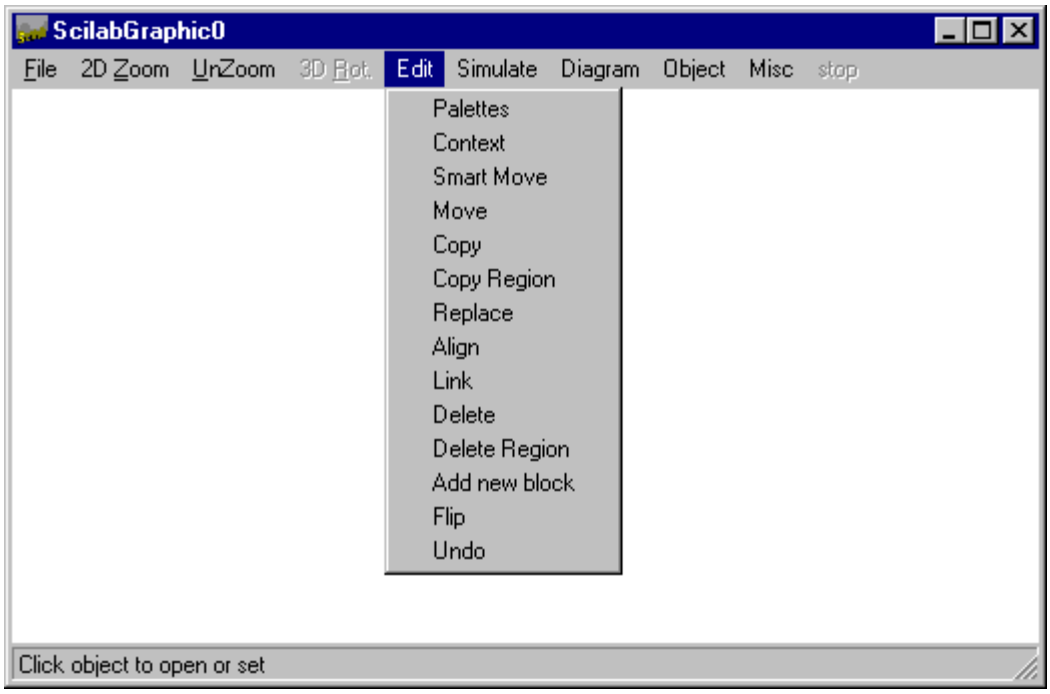


2.1. Construcción de un Modelo con Scicos.

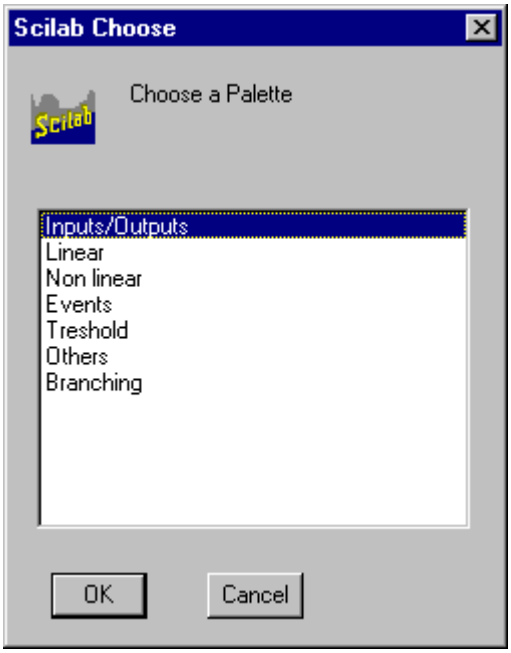
La construcción de un modelo en Scicos consiste típicamente en :

2.1.a. Abrir Paleletas de Bloques.

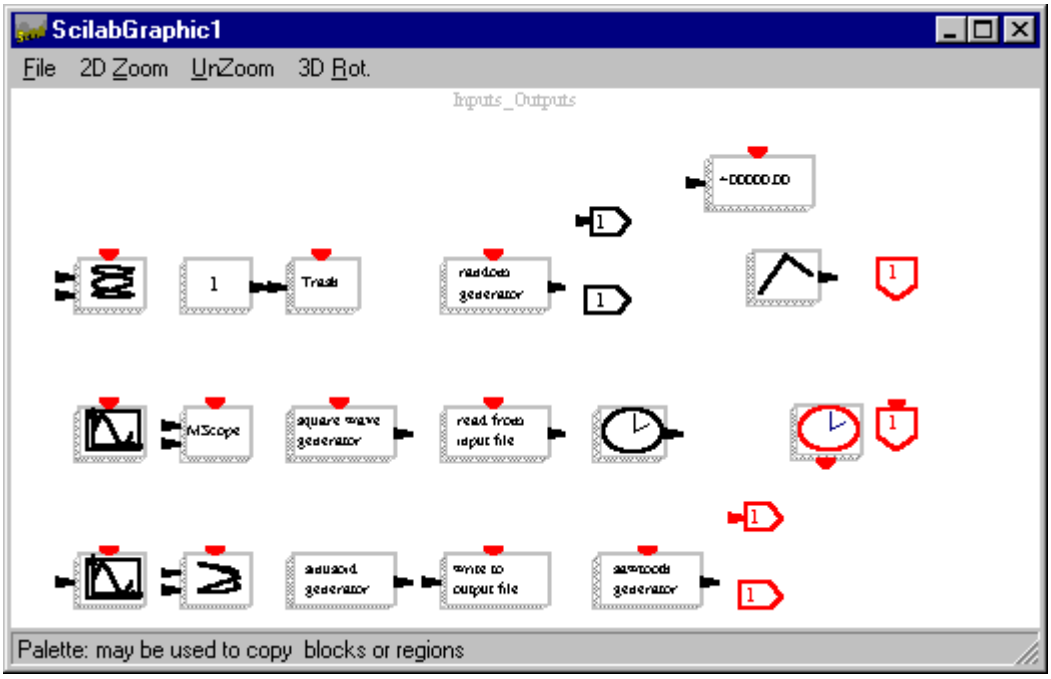
El primer paso para construir un nuevo modelo es abrir una o más paletas de bloques, usando el botón **Palettes** que se encuentra dentro del menú **Edit** :



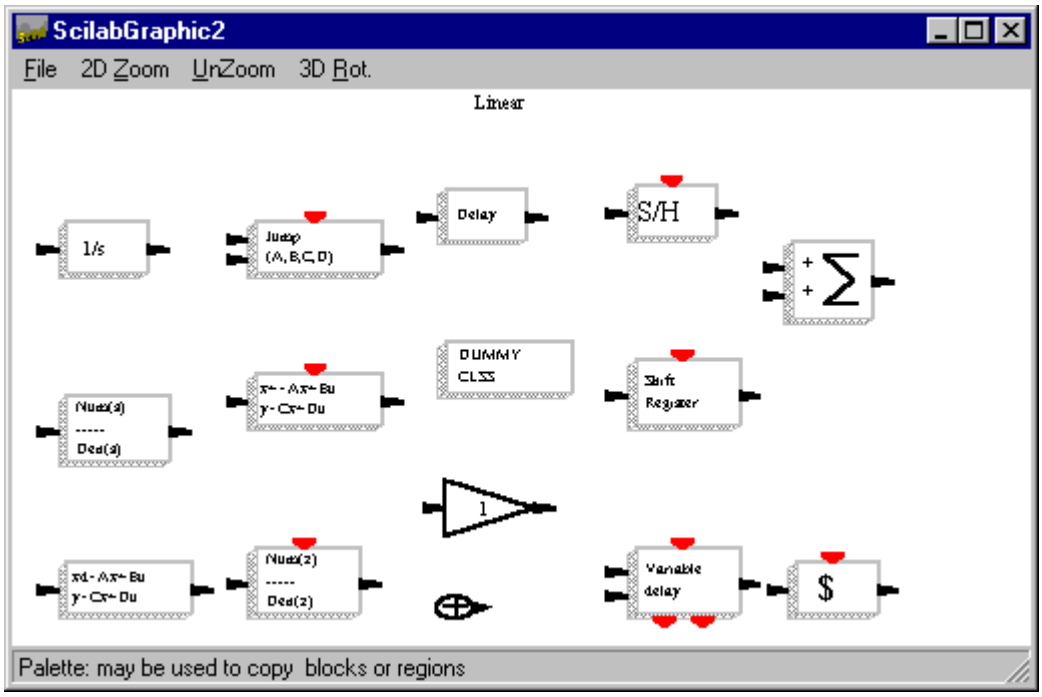
Apareciendo la siguiente ventana de dialogo , donde puedo elegir la paleta deseada:



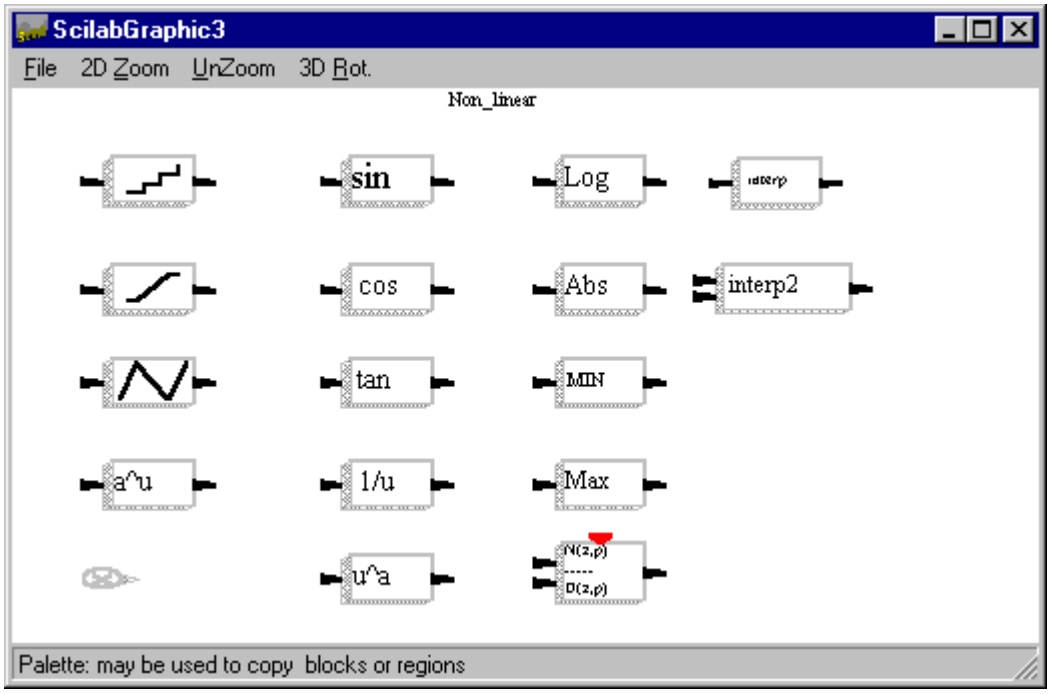
- ***Paleta de Entradas/ Salidas*** (Inputs/Outputs)



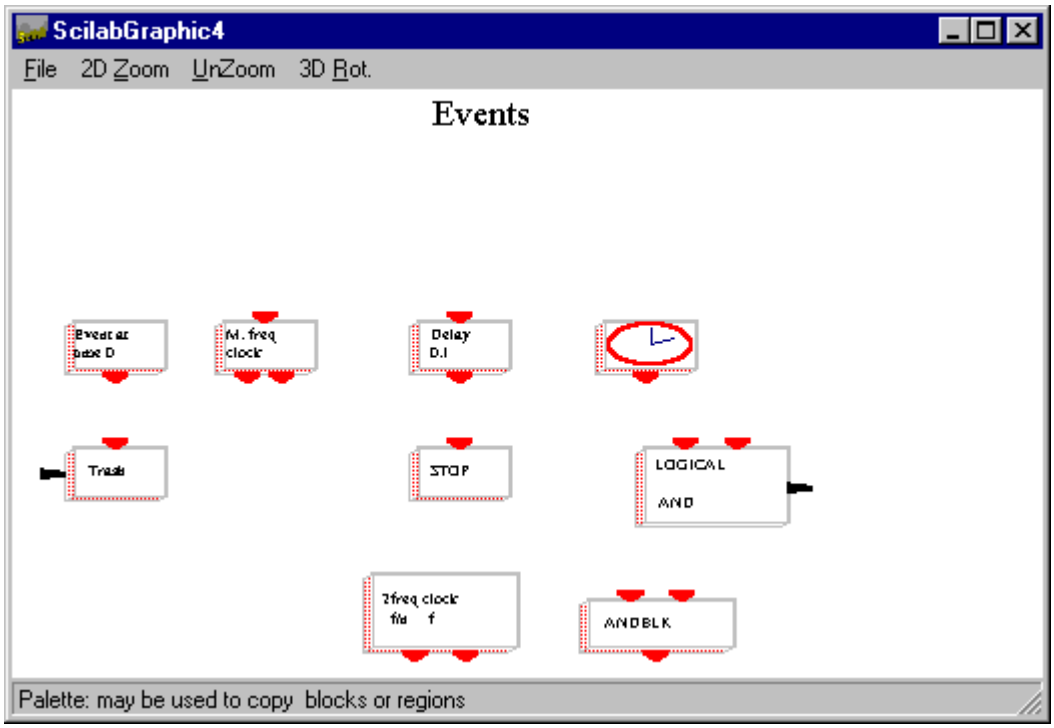
- ***Paleta Lineal*** (Linear)



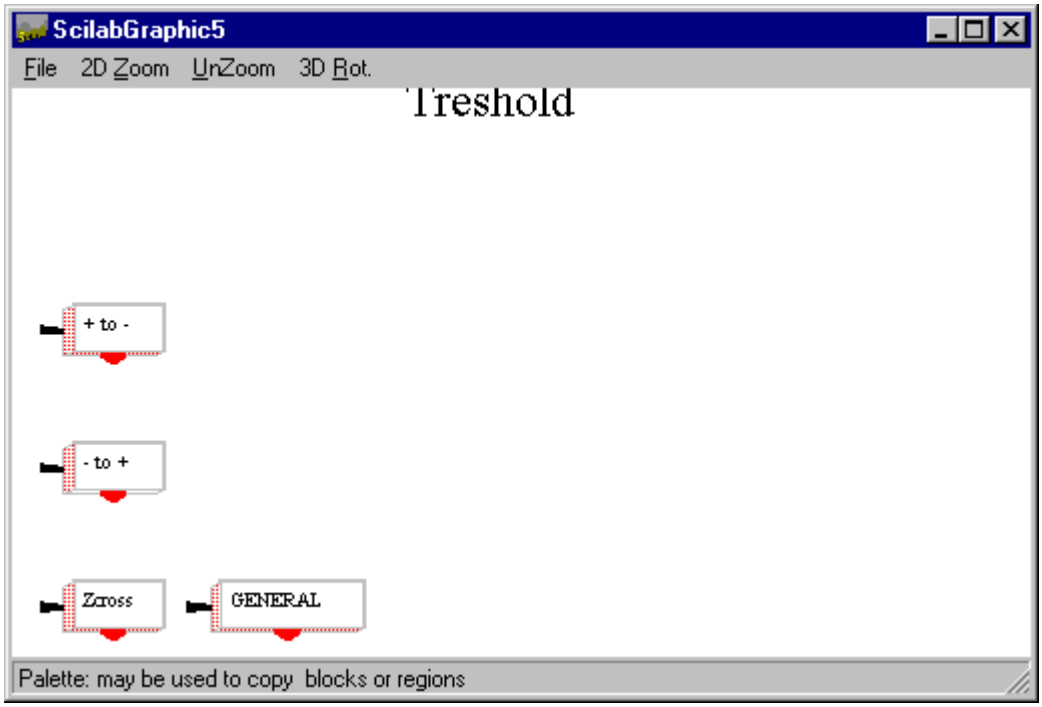
- ***Paleta No Lineal*** (Non Linear)



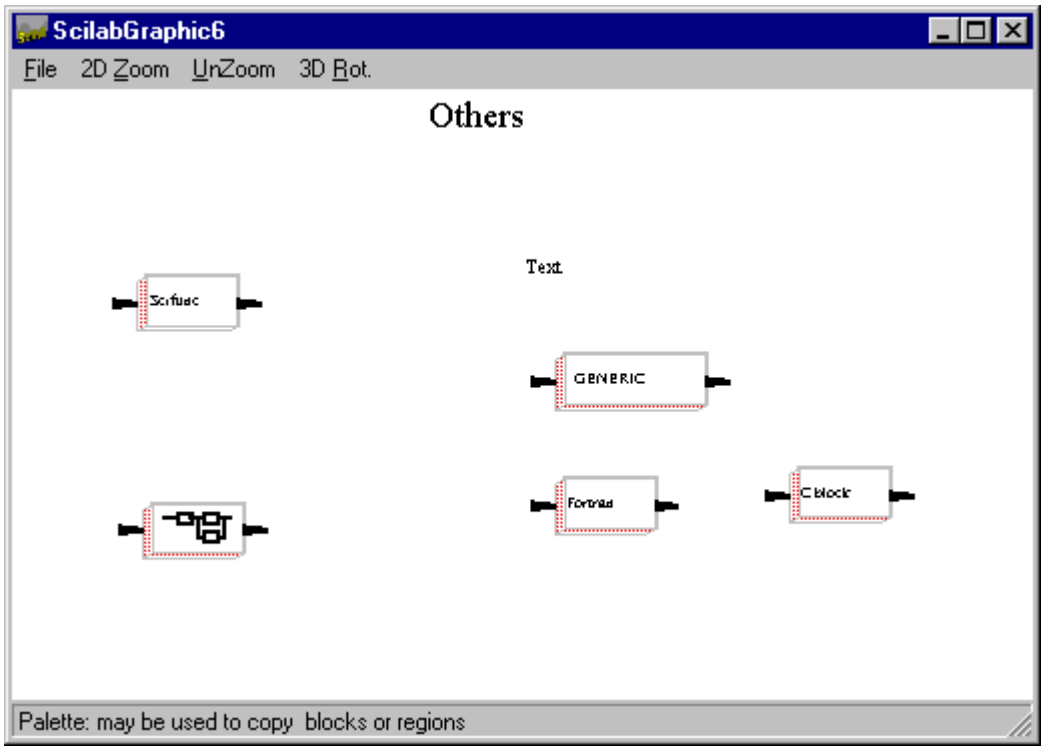
- ***Paleta de Eventos*** (Events)



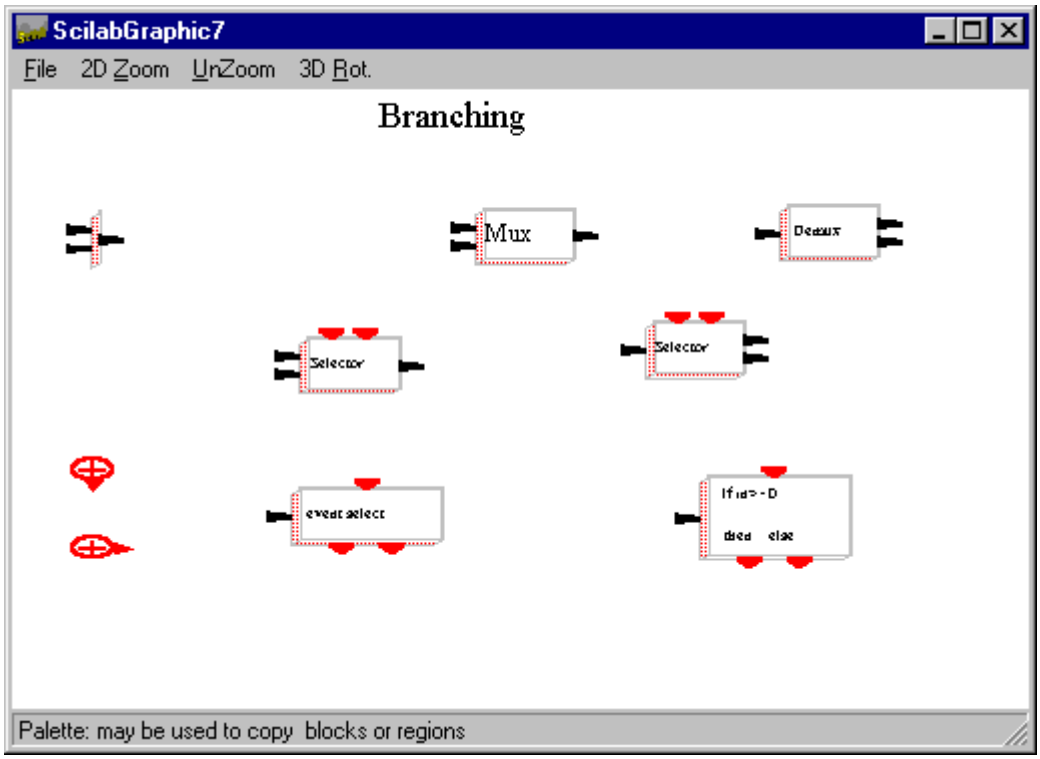
- ***Paleta de Treshold***



- ***Paleta Otros*** (Others)

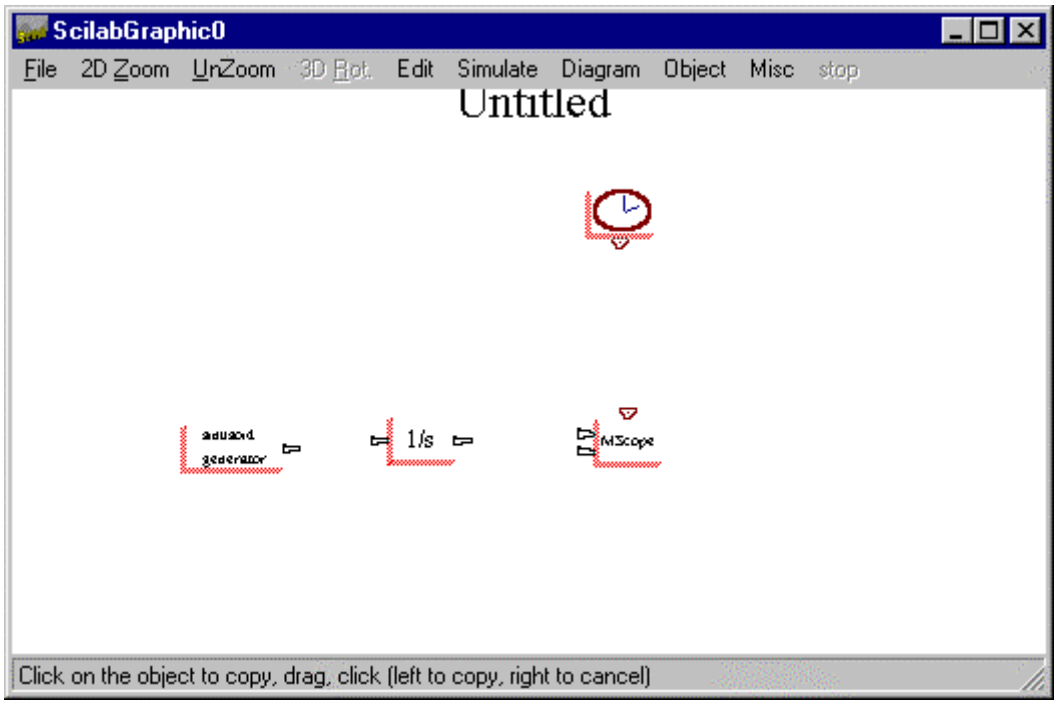


- ***Paleta de Bifurcación*** (Branching)



2.1.b. Copia de Bloques.

La copia de bloques en la ventana principal de Scicos desde las paletas, se realiza haciendo click en el botón **copy** del menú **Edit**. Vamos a construir ***un modelo como ejemplo***, para copiar hago click con el botón izquierdo del ratón y arrastro hasta la ventana principal de Scicos; volviendo hacer click con el botón izquierdo. Obteniendo para nuestro modelo en construcción la siguiente ventana:

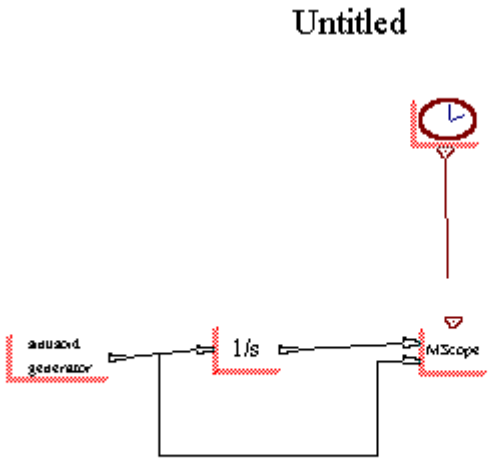


Con el botón derecho del ratón cancelo la selección de bloque realizada, además si quiero borrar un bloque ya pegado en la ventana principal, hago click en el botón **Delete** del menú **Edit** y hago click sobre el bloque a borrar.

2.1.c. Conexión de Bloques. (*link*)

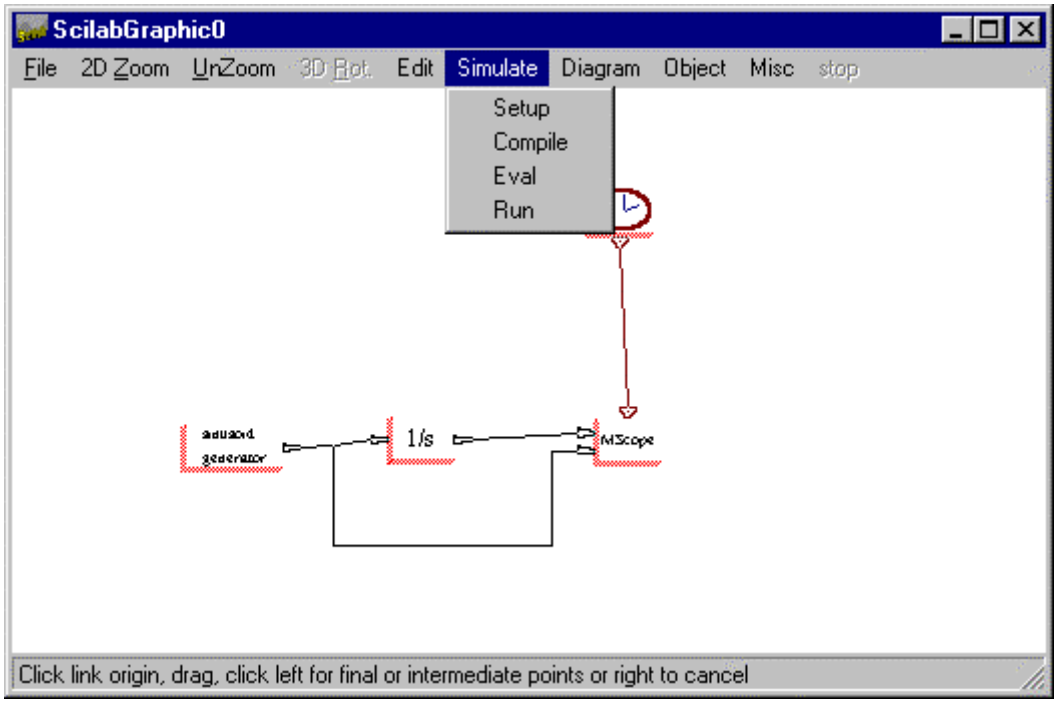
Debo conectar las entradas y salidas de los bloques, seleccionando primero el botón **Link** del menu **Edit**. Una vez he hecho click en el botón **Link**, hago click sobre la salida de un bloque y luego sobre la entrada del siguiente, quedando ambos conectados. También puedo partir o terminar en puntos intermedios en los enlaces, o sea hago click sobre el botón **Link** y pincho en un punto de un enlace a partir del cual puedo obtener las bifurcaciones que quiera.

El proceso de enlazado puede ser parada cuando quiera simplemente haciendo click en el botón de la derecha del ratón. Es importante reseñar que es muy importante incluir en cualquier modelo Scicos un bloque **scope** o **write to file**, para poder visualizar los datos de una simulación .



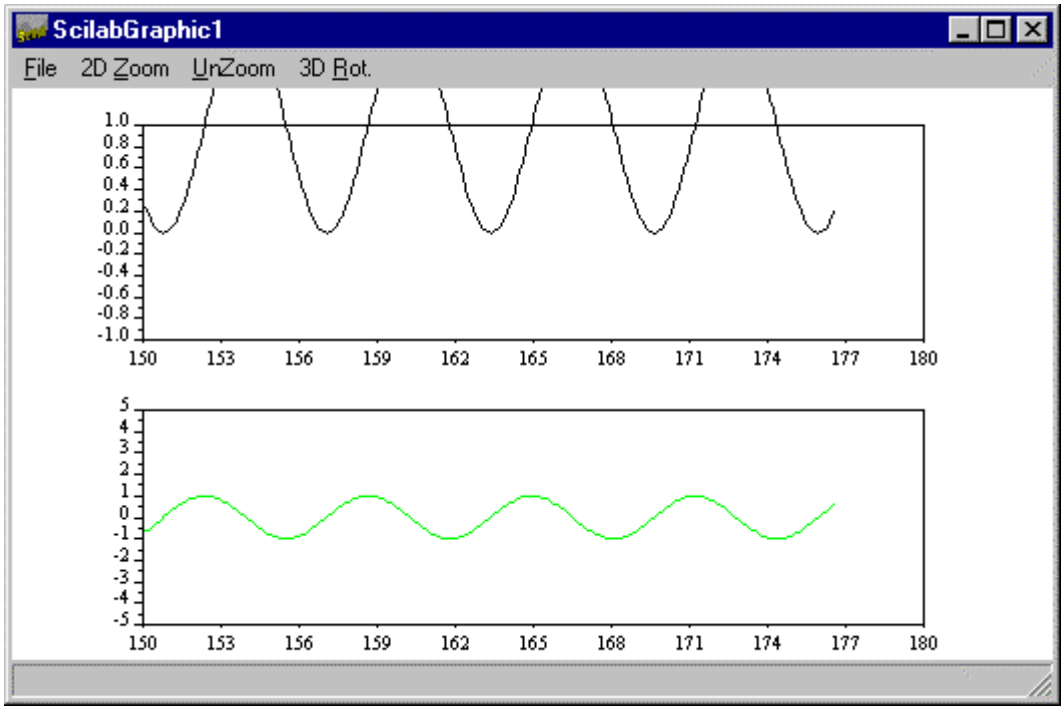
2.2. Simulación de un Modelo con Scicos.

Una vez que tengo completo el modelo puedo realizar una simulación usando **Run** del menú **Simulate**.



Una vez seleccionado el botón **Run** se realiza la compilación del modelo (si no ha sido ya compilado) y después la simulación. La *simulación* puede ser parada en cualquier momento simplemente haciendo click sobre el botón **Stop** en la ventana principal de Scicos.

El ***resultado de la simulación*** de este ejemplo se muestra en la siguiente ventana :

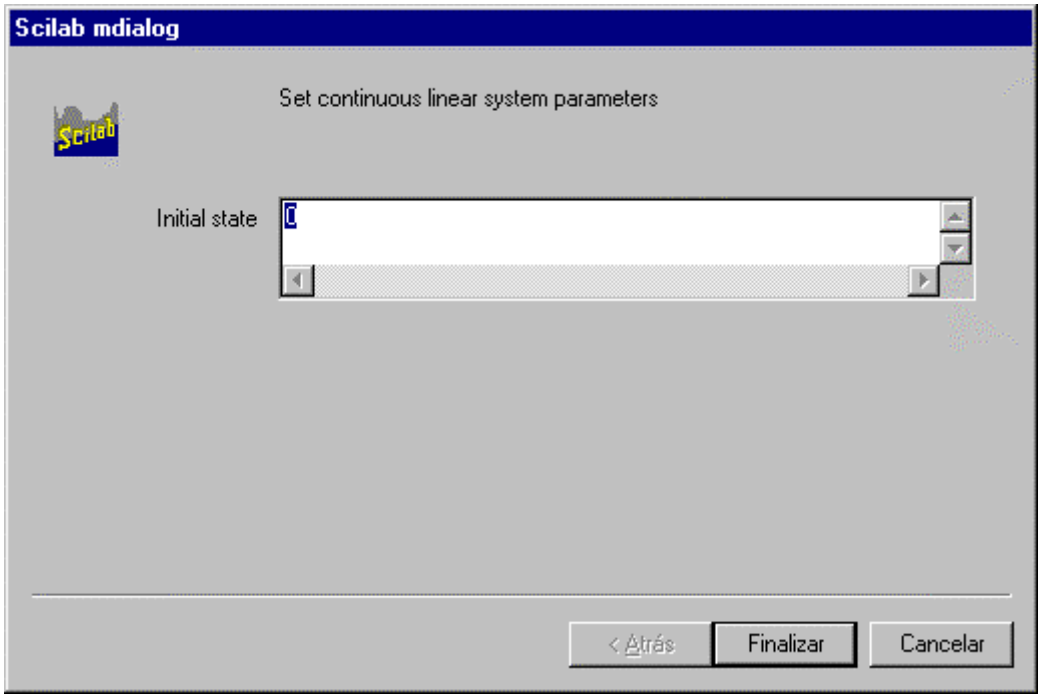


Un modelo compilado puede ser salvado como un fichero *.cos.

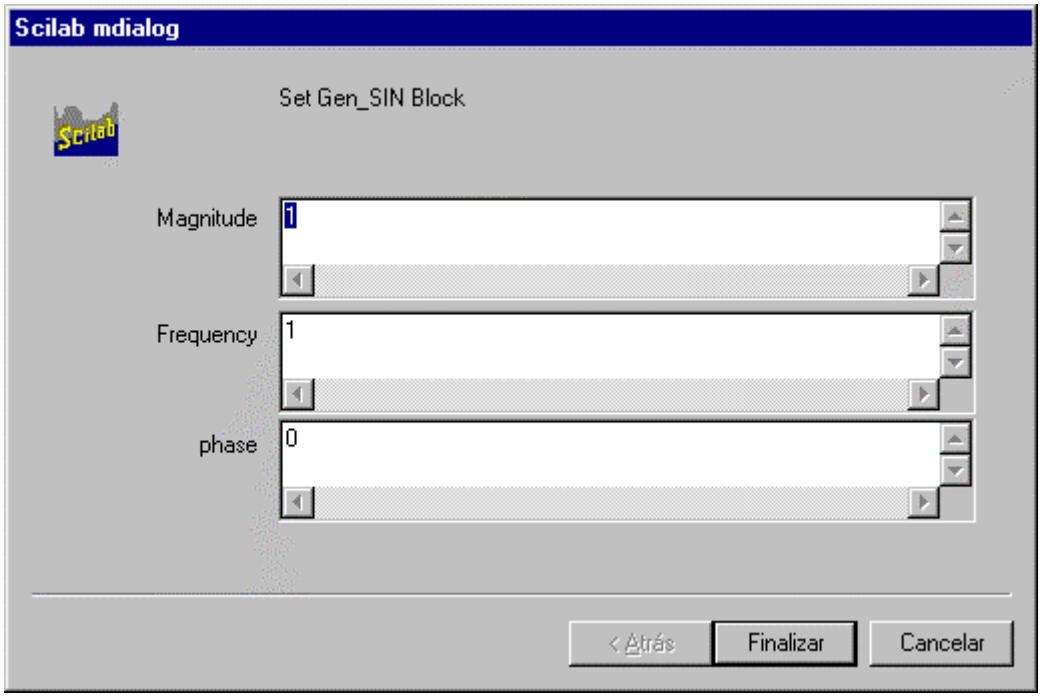
2.3. Modificación de los Parámetros de los Bloques.

Como podemos ver en las gráficas anteriores se ha obtenido una simulación con los ***parámetros*** que incluían los bloques por defecto. Con una simple ojeada se puede observar que las escalas no están acordes con las señales, por lo que los parámetros de los bloques pueden ser modificados abriendo los menús de dialogo, simplemente seleccionando el botón **Open/set** del menú **Object**. Algunas veces no sólo querré resolver un problema de escalas, si no que a lo mejor quiero simular un sistema dinámico para los que quiero obtener diferentes simulaciones de su ecuación diferencial con distintos coeficientes cada vez. Pero ahora vamos a ***modificar los parámetros*** de los bloques de nuestro modelo ejemplo, con el único fin de obtener una salida gráfica en condiciones.

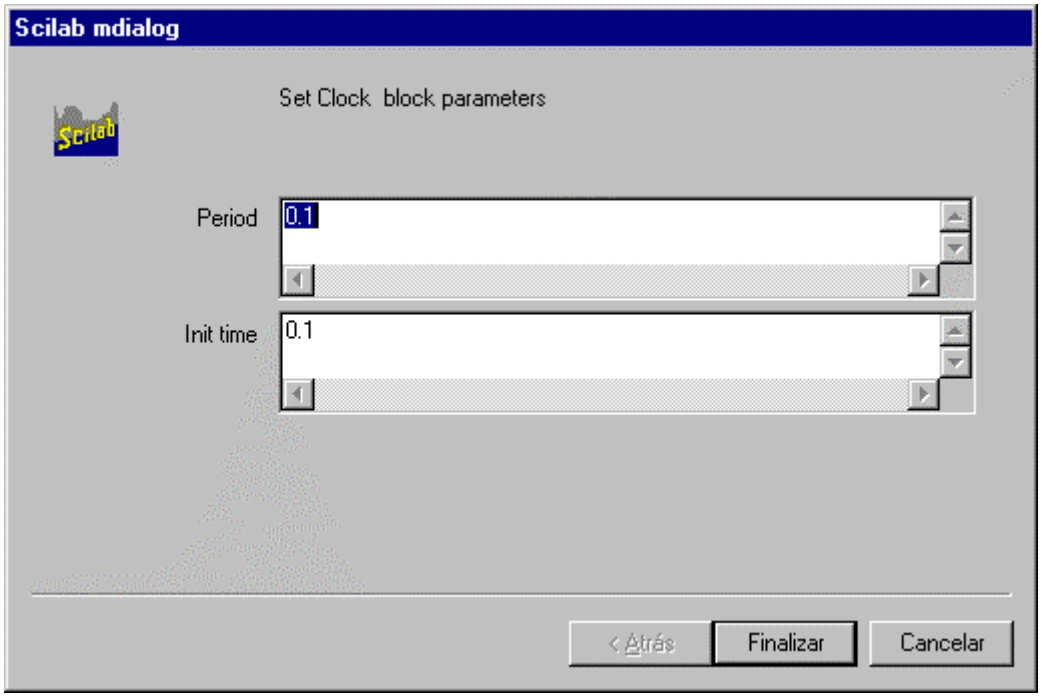
El siguiente menú de dialogo corresponde al **sistema lineal**, que no voy a modificar sus parámetros:



El siguiente menú de dialogo corresponde al **Generador Sinusoidal**, que no voy a modificar sus parámetros:

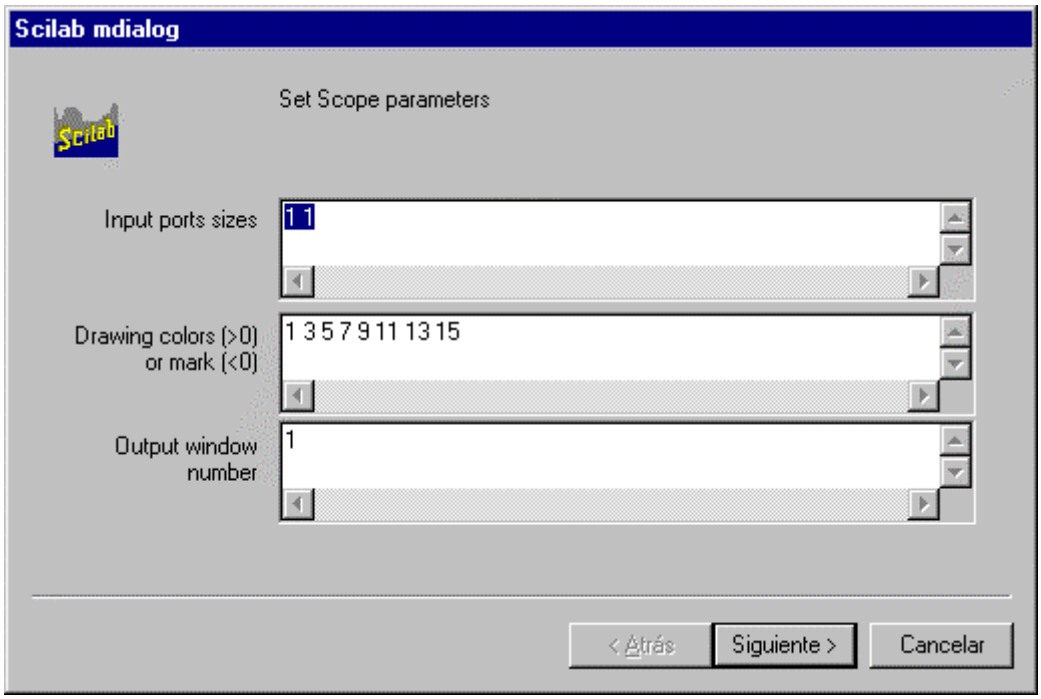


El siguiente menú de dialogo corresponde al **Reloj**, que no voy a modificar sus parámetros:

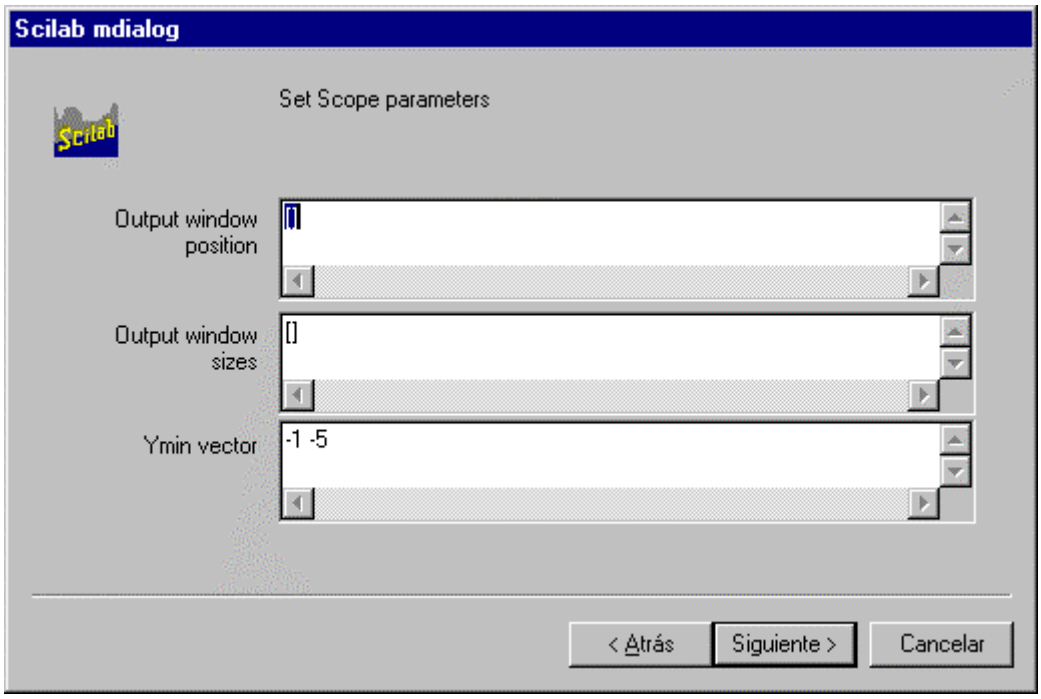


Los siguiente menús de dialogo corresponden al bloque **MScope**, que si voy a modificar sus parámetros:

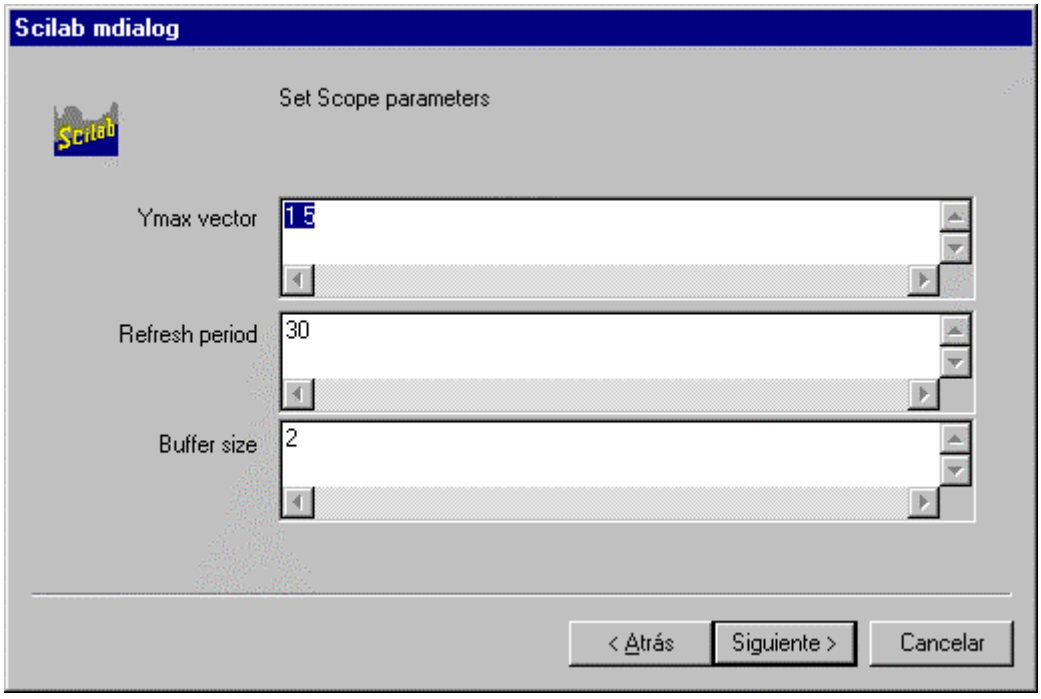
- En esta ventana introduzco en **Drawing colors** = 5 3 5 7 9 11 13 15



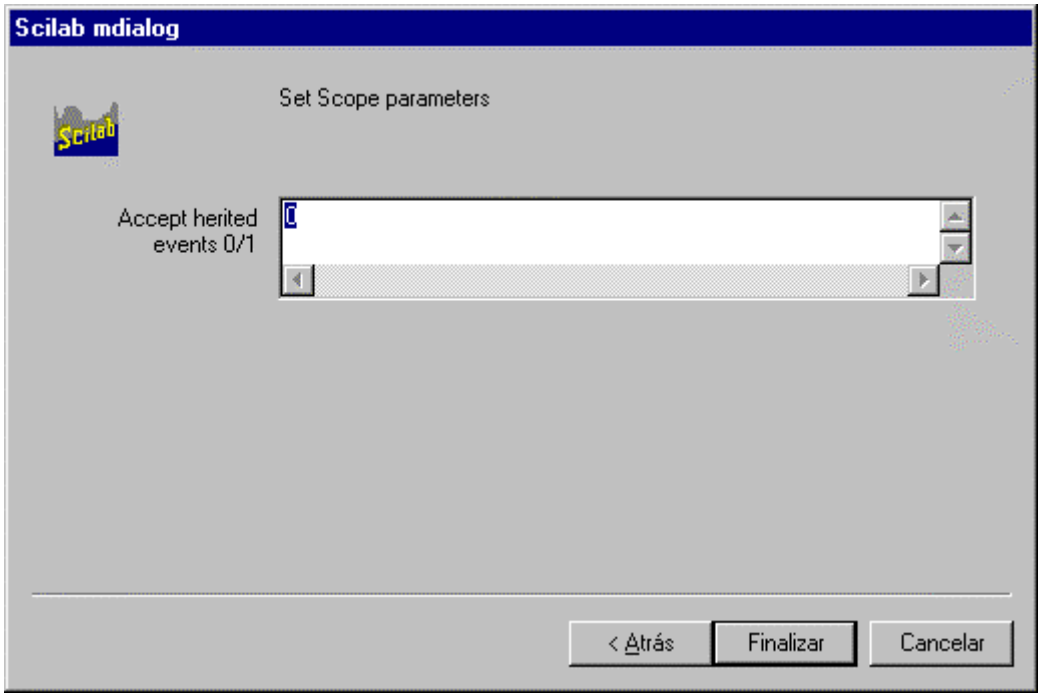
- En esta ventana introduzco en **Ymin vector** = 0 –1



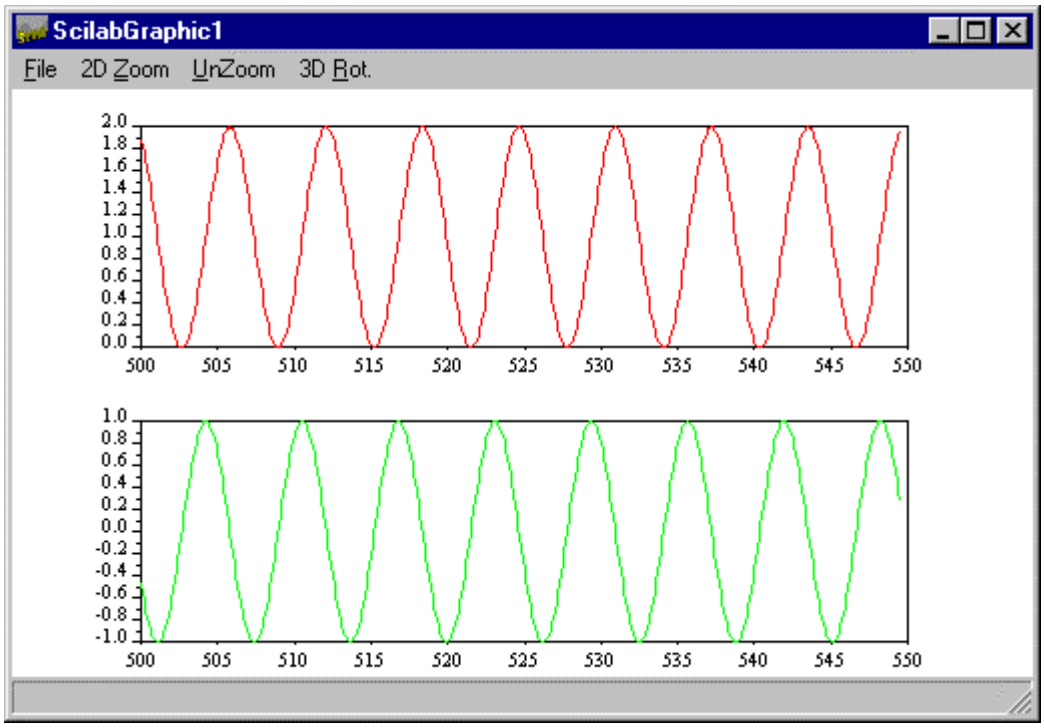
- En esta ventana introduzco en **Ymax vector** = 2 1, **Refresh period** = 50 y **Buffer size** = 10.



- Esta ventana no la toco.



A continuación *vuelvo a simular* el modelo, obteniéndose el siguiente resultado después de las modificaciones :



Estos parámetros pueden ser definidos utilizando expresiones Scilab e incluso puedo utilizar variables de Scilab si ya han sido definidas. Estas expresiones son memorizadas simbólicamente y evaluadas posteriormente.

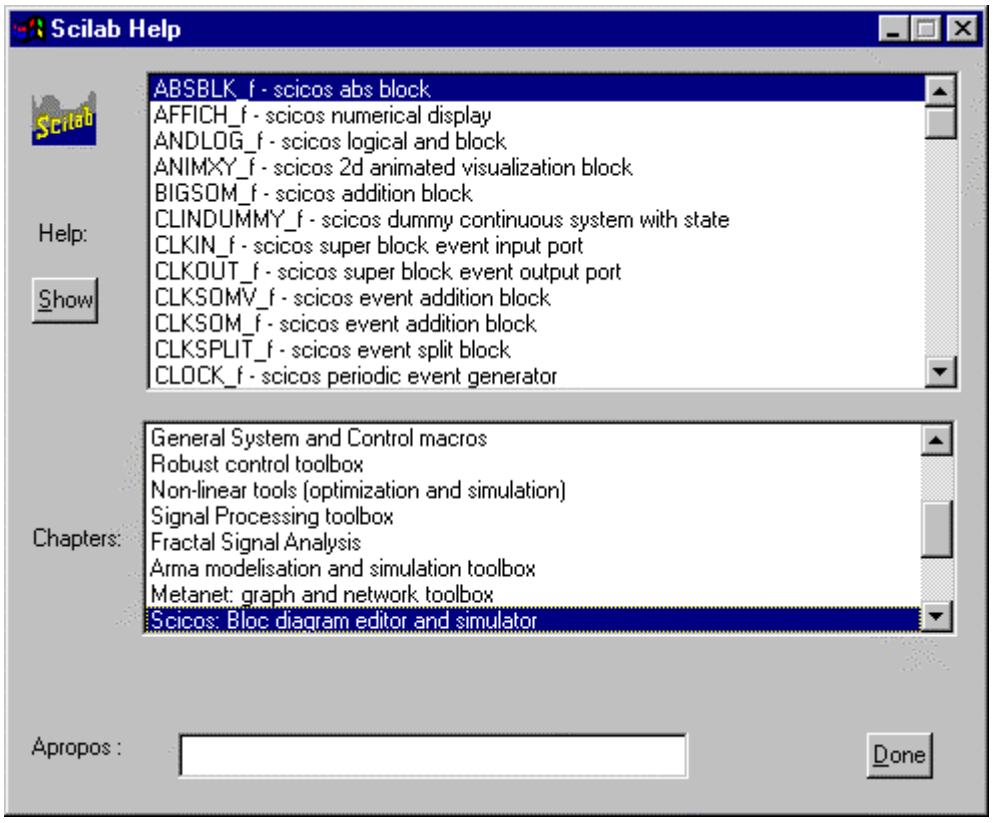
2.4. Otras Funcionalidades de Scicos.

El esitor Scicos proporciona muchas otras *funcionalidades* tales como :

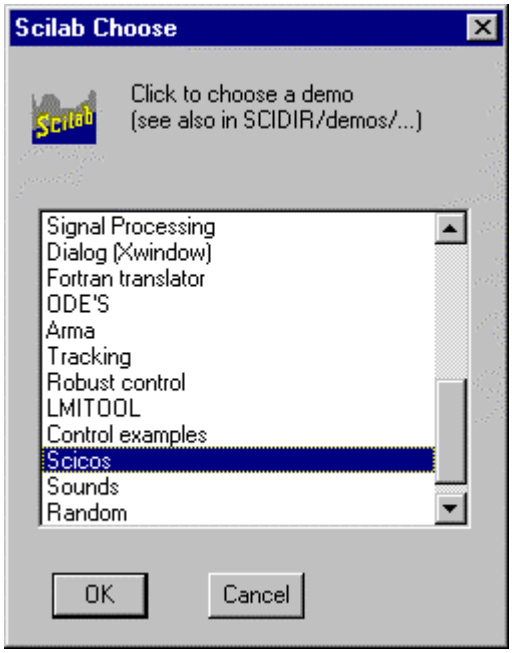
- Salvar y cargar diagramas de modelos en varios formatos.
- Ampliación de zonas del modelo.
- Cambio del aspecto de los bloques y colores.
- Cambio del color de fondo de los diagramas de modelos y de los bloques.
- Ubicar texto en el diagrama.
- Impresión y exportación de diagramas Scicos.
- Y muchas otras funciones GUI estándar .

El botón de **ayuda** de la ventana principal de Scicos puede usarse de dos maneras : seleccionando el botón **ayuda** y haciendo click sobre un bloque obtengo un manual de ayuda del bloque o seleccionando el botón **ayuda** y haciendo click sobre otro botón, mostrándose el manual de ayuda de ese botón.

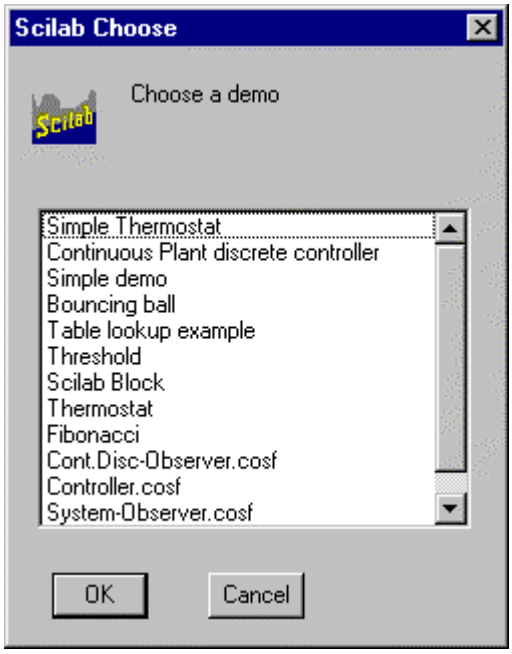
También puedo obtener información de las funciones que puedo implementar en la programación de nuevos bloques desde la ventana principal de Scillab haciendo click en el botón **Help Dialog** del menú **Help** apareciendo, la siguiente ventana en la que elijo Scicos en zona inferior **Chapters** y seleccionando en la zona superior la función de la que quiera ayuda y pulsando el botón **Show** :



Si quiero ejecutar las **demos** de Scicos selecciono el botón **Demos** del menú **File** en la ventana principal de Scilab, apareciendo la siguiente ventana de diálogo :

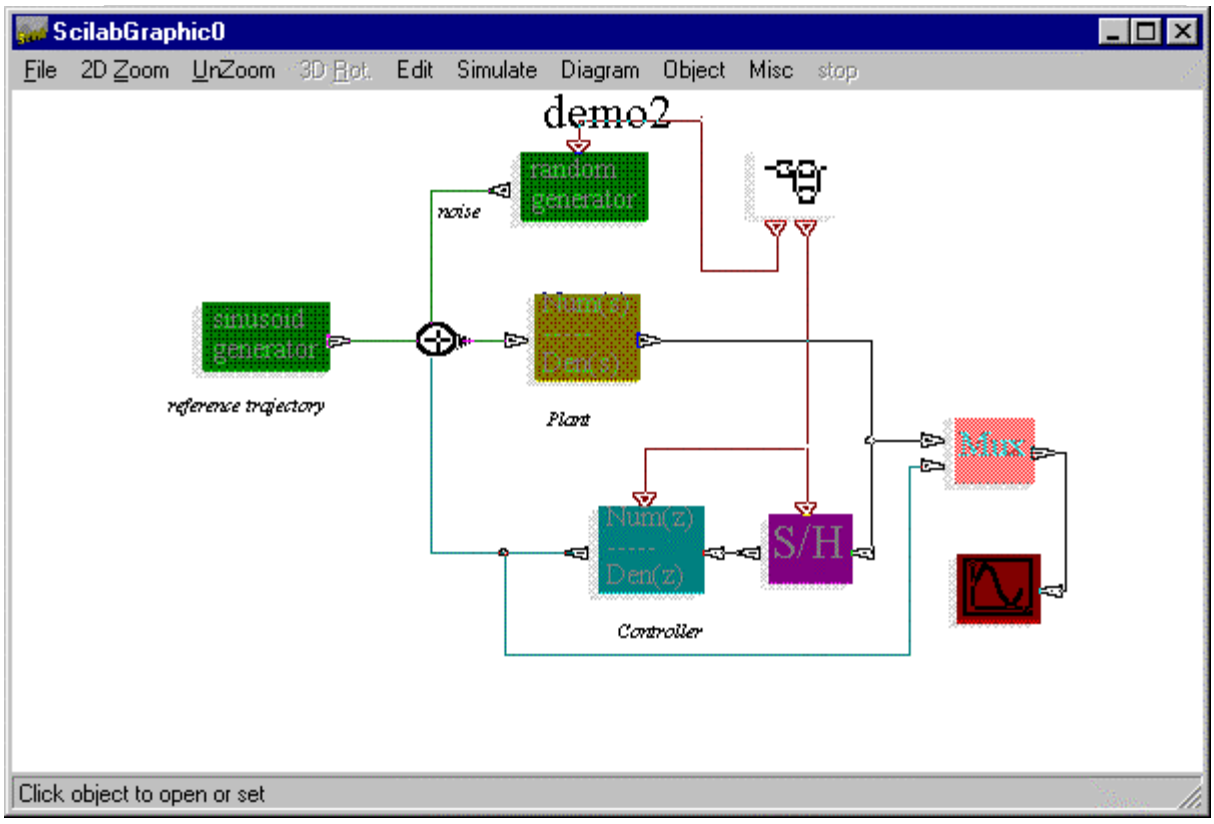


Selecciono Scicos y hago click en el botón **OK**, apareciendo otra ventana con todas las demos disponibles para Scicos, seleccionando la que quiera ejecutar y haciendo click en el botón **OK** :

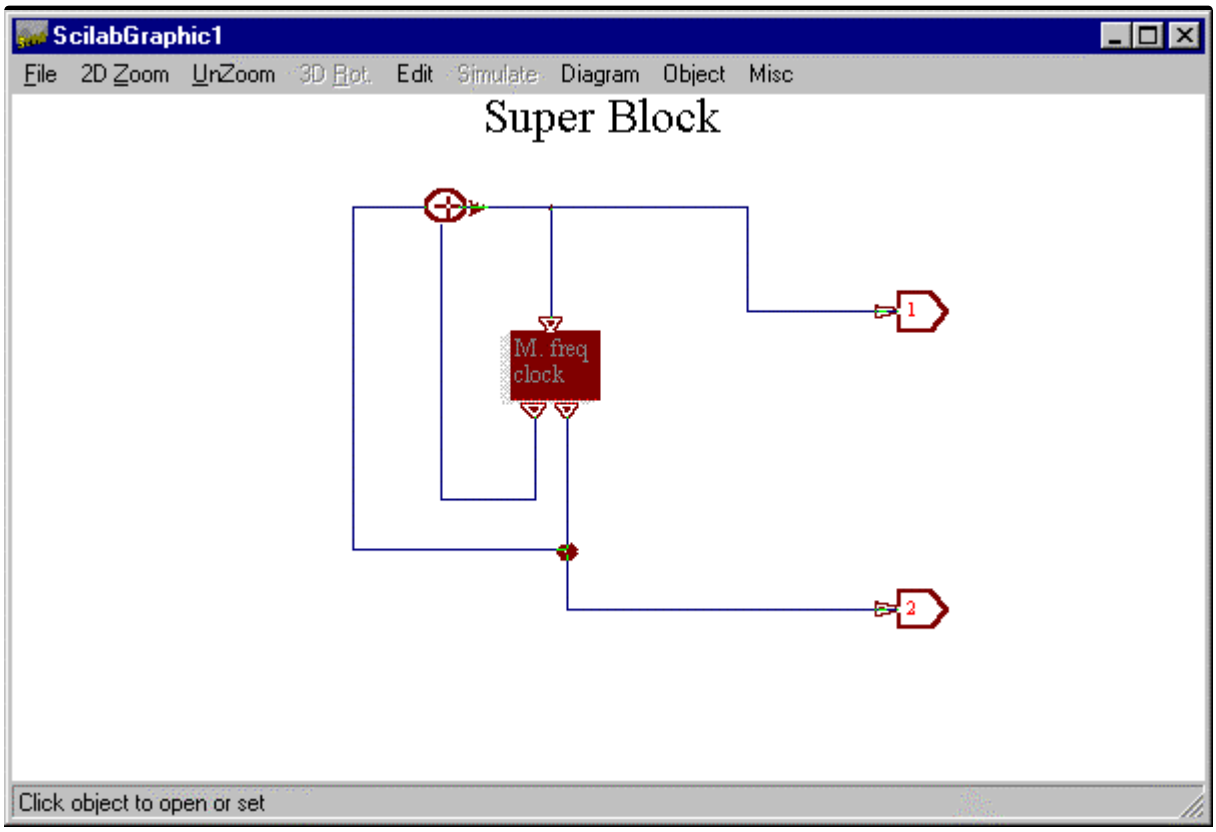


Finalmente, una importante característica en Scicos es la posibilidad de crear *subsistemas* . No es deseable tener modelos complejos en Scicos con cientos de bloques en un solo diagrama, por lo que Scicos ofrece la posibilidad de hacer agrupaciones de bloques. Para esto Scicos proporciona la posibilidad de agrupamiento de bloques definiendo subdiagramas llamados **Super Bloques** (*Super Blocks*), que se comportan como cualquier otro bloque pero pueden contener un número ilimitado de bloques, e incluso otros **super bloques**. Los **super bloques** tienen la siguiente apariencia :

Que en uno de los ejemplos de las ***demos*** podemos verlo en el diagrama del modelo siguiente:



Si abrimos el *super bloque* haciendo doble click sobre el icono podemos observar todos los bloques que contiene :



3. TIPOS DE BLOQUES EN SCICOS.

Hay tres tipos de bloques fundamentales en Scicos : **Bloque normal** (*Regular Block*), **Bloque de Cruce por Cero** (*Zero Crossing Block*) y **Bloque sincronizado** (*Synchro Block*). Estos bloques pueden tener dos tipos de entradas y dos tipos de salidas: entradas normales, entradas de activación, salidas normales y salidas de activación. Las entradas y salidas normales se conectan mediante **enlaces** (*link*) **normales** , mientras que las entradas y salidas de activación mediante enlaces de activación. Los *puertos de las entradas de activación* se encuentran en la parte superior del bloque, mientras que los *puertos de las salidas de activación* se encuentran en la parte inferior.

3.1. Bloques Normales Simples.

3.1.a. Bloque Normal Simple Continuo.

Los *bloques normales simples continuos* pueden tener un *estado* ***continuo x*** y un *estado* ***discreto z***. Si consideramos que tiene un **estado continuo x** y si **u** representa su entrada normal, entonces cuando el bloque está activo durante un intervalo de tiempo, su estado x evoluciona de acuerdo con :

$$\dot{x} = f(t, x, z, u, p, n_e)$$

Donde ***f*** es un vector función, ***p*** un vector de parámetros constantes y n_e que es la clave de activación (*activation code*) representada por un entero designando el puerto(s) a través del cual es activado. En el caso en que las *entradas de activación* sean i_1 , i_2,.....,i_n entonces :

$$n_e = \sum_{j=1}^{n} 2^{i_j - 1}.$$

Por otro lado si el bloque estuviera activado por un evento, los estados x y z cambian instantáneamente de acuerdo con las siguientes ecuaciones:

$$\begin{aligned} x(t_e) &= g_c(t_e, x(t_e^-), z(t_e^-), u(t_e), p, n_e) \\ z(t_e) &= g_d(t_e, x(t_e^-), z(t_e^-), u(t_e), p, n_e) \end{aligned}$$

donde t_e representa el tiempo en el que se produce el evento y el estado discreto z permanece constante entre dos eventos sucesivos, luego $z(t_e^-)$ puede ser interpretado como el valor previo de z. Durante *el tiempo de activación*, la salida normal del bloque viene definida por :

$$y(t) = h(t, x(t^-), z(t^-), u(t), p, n_e)$$

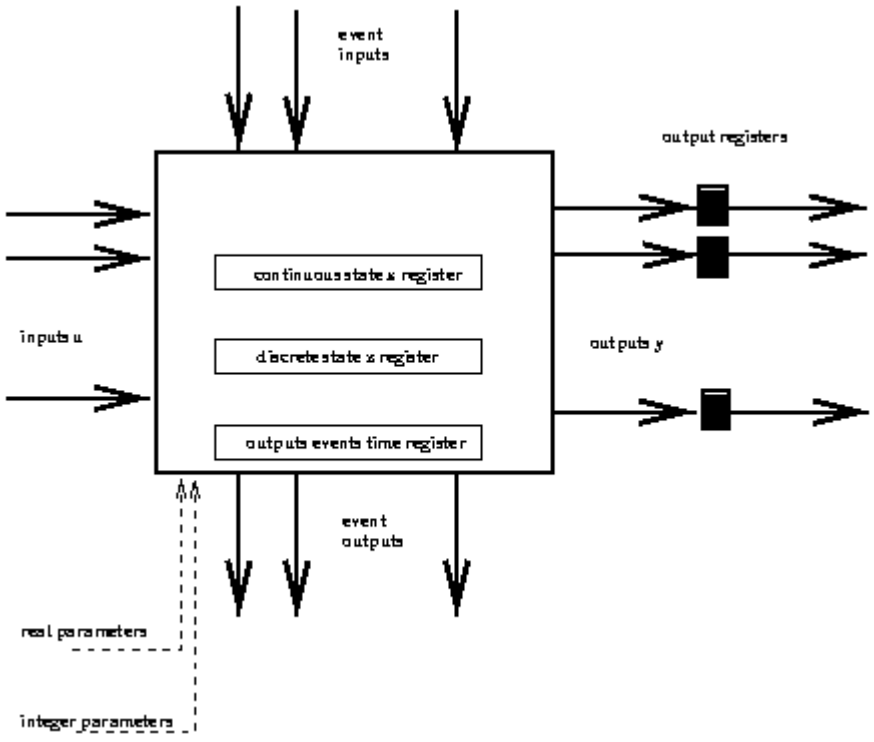
que será constante cuando el bloque no esté activo.

Este tipo de bloques pueden generar señales de activación de tipo evento . Si esta señal es activada por un evento en el instante t_e , entonces el tiempo de cada **salida tipo evento** (*output event*) viene dada por:

$$t_{evo} = k(t_e, z(t_e), u(t_e), p, n_e)$$

donde t_{evo} es un vector de tiempo. La generación de eventos puede ser preprogramada asociando las variables de disparo de los bloques con la salida de eventos.

A continuación se muestra un figura resumen de un ***bloque normal simple continuo*** :



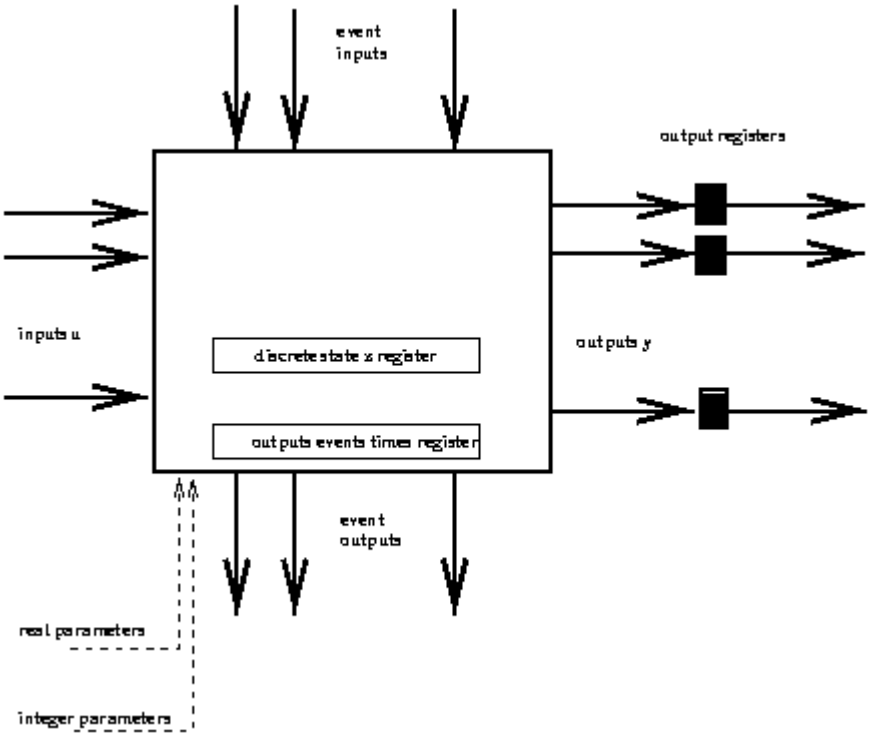
3.1.b. Bloque Normal Simple Discreto.

Los bloques normales continuos están constantemente actualizando sus salidas y su estado continuo . Mientras que los ***bloques normales simples discretos*** actúan solo cuando reciben un evento de entrada y sus acciones son instantáneas, por lo que deben tener una entrada de evento mínimo. Si ***u*** es una señal de entrada e ***y*** la de salida, con la llegada de un evento (o eventos) en un tiempo t_e , entonces su estado y sus salidas cambiarán de la siguiente forma:

$$z := f_d(t_e, z(t_e^-), u(t_e^-), p, n_{evprt})$$
$$y := h_d(t_e, z, u(t_e), p)$$

donde f_d y h_d son funciones específicas del bloque, *p* es un vector de parámetros constante y n_{evprt} designa el puerto (s) a través del cual el evento(s) ha (han) llegado. No es necesario decir que la *y* permanece constante entre dos sucesivos eventos. La diferencia entre los **bloques continuos** y los **discretos** es que estos últimos no pueden tener un estado continuo y sus salidas permanecen constantes entre dos eventos.

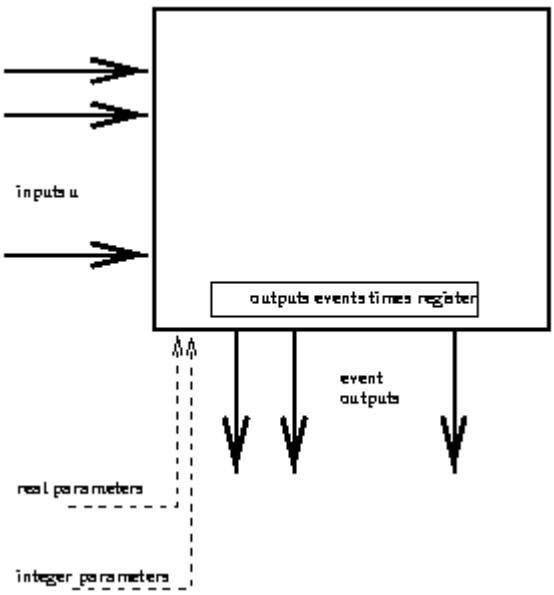
A continuación se muestra una figura resumen de un ***bloque normal simple discreto*** :



3.2. Bloque Simple de Cruce por Cero.

Los ***bloques simples de cruce por cero*** tienen entradas de señales normales, entradas y salidas de eventos pero no tienen posibilidad de señales de salida normales. Un *bloque de cruce por cero* puede generar un evento de salida solamente si al menos una de sus entradas normales cruza el cero (cambia de signo). En ese caso, se produce un evento, y su tiempo de activación va a depender de que señales de entrada pasen por cero y del signo de las ellas justo cuando haya ocurrido el cruce. Existen ejemplos de este tipo de bloques en la **Paleta Umbral** (*Threshold Palette*).

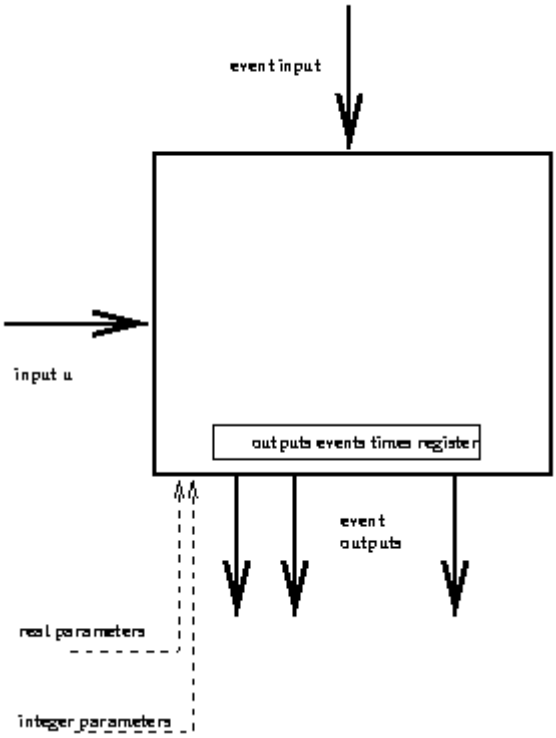
A continuación se muestra una figura resumen de un ***bloque de Cruce por Cero*** :

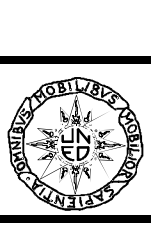


3.3. Bloque Simple Sincronizado.

Los ***bloques simples sincronizados*** generan señales de salida de activación que están sincronizadas con sus señales de entrada de activación. Estos bloques tienen una única entrada de activación, que es enrutada a una de sus salidas de activación. La elección de una de estas salidas depende del valor de la señal de entrada normal. Podemos de este tipo de bloques en la **paleta** (*branching palette*) como son los *bloques* **If-then-else** y **event select**.

A continuación se muestra una figura resumen de un ***bloque simple sincronizado*** :





4. HERENCIA Y DEPENDENCIA TEMPORAL.

Para evitar que se tengan que dibujar todas las señales de activación en un diagrama Scicos de un modelo, Scicos utiliza una característica llamada ***herencia*** . Si un bloque no tiene una entrada de activación, este hereda la señal de activación de la señal de entrada normal. Y para los bloques que están activos permanentemente, se les puede declarar como **dependientes temporalmente** (*time dependent*) y no siendo necesario de esta forma entradas de activación. Es importante reseñar que los bloques con **dependencia temporal** no heredan.

5. CONSTRUCCIÓN DE NUEVOS BLOQUES.

Un nuevo bloque puede ser construido como un ***Super Bloque*** (mediante la interconexión de bloques simples) compilandose posteriormente. Pero para construir un nuevo ***Bloque Simple*** tiene que ser mediante dos funciones :

- **Función Interfaz** (*Interfacing function*) que determina el interfaz con el usuario. La función interfaz está siempre escrita en código Scilab, pudiendose utilizar funciones Scilab o que sean desarrolladas por el usuario.

- **Función Computacional** (*Computacional function*) para especificar el comportamiento dinámico del bloque. La función computacional puede ser programada en *C , Fortran* y en *lenguaje Scilab*. Las rutinas de *C* y *Fortran* son enlazados dinámicamente, pero con rutinas Scilab interactúa permanentemente dando mejores resultados en lo que concierne al rendimiento de la simulación.

5.1. Función Interfaz.

La ***función interfaz*** determina la geometría, color, número de puertos, tamaño, icono, etc. En adición con los estado inicial y los parámetros. Esta función también proporciona el dialogo del usuario con el bloque. En cuanto a lo que la función interfaz debería hacer y retornar depende de la ***variable de entrada de status*** (input flag) **job.**

5.1.a. Sintaxis .

[x , y , typ] = block (job, arg1, arg2)

Parámetros :

- **job** ==’**plot** ‘ : la función dibuja el bloque.

  arg1 es la estructura de datos del bloque.
  arg2 no se usa.
  x, y , typ no se usa.

  En general, podemos usar la función **standard_draw** la cual dibuja un bloque rectangular, y los puertos de entrada y de salida. También determina el tamaño, icono y color que dan el aspecto al bloque.

- **job** ==’**getinputs**‘ : la función retorna la posición y el tipo de puertos de entrada (normal o activación).

  arg1 es la estructura de datos del bloque.
  arg2 no se usa.
  x es el vector de las coordenadas x de los puertos de entrada.

y es el vector de las coordenadas y de los puertos de entrada.
typ es el vector del tipo de puertos de entrada (1 para regular y 2 para activación)

En general, podemos usar la función **standard_input** .

- **job** ==’**getoutput**‘ : la función retorna la posición y el tipo de puertos de salida (normal o activación).

  arg1 es la estructura de datos del bloque.
  arg2 no se usa.
  x es el vector de las coordenadas x de los puertos de salida.
  y es el vector de las coordenadas y de los puertos de salida.
  typ es el vector del tipo de puertos de salida (1 para regular y 2 para activación)

  En general, podemos usar la función **standard_output** .

- **job** ==’**getorigin**‘ : la función retorna las coordenadas del punto inferior izquierdo del rectángulo que contiene la silueta del bloque.

  arg1 es la estructura de datos del bloque.
  arg2 no se usa.
  x es el vector de la coordenadas x del punto inferior izquierdo de el bloque.
  y es el vector de la coordenadas y del punto inferior izquierdo de el bloque.
  typ no se usa.

  En general, podemos usar la función **standard_origin** .

- **job** ==’**set**‘ : la función abre un dialogo para adquirir los parámetros del bloque.

  arg1 es la estructura de datos del bloque.
  arg2 no se usa.
  x es la nueva estructura de datos del bloque.
  y no se usa.
  typ no se usa.

  En general, podemos usar la función **standard_origin** .

- **job** ==’**define**‘ :Inicialización de la estructura de datos del bloque (nombre de la función computacional, tipo, número y tamaño de las entradas y salidas, etc.)

  arg1 no se usa.
  arg2 no se usa.
  x es la estructura de datos del bloque.
  y no se usa.
  typ no se usa.

5.1.b. Definición de la Estructura de Datos del Bloque.

Cada bloque Scicos está definido por una **estructura de datos Scilab** como sigue :

list (‘ Block ‘ , graphics, model, unused, GUI_function)

Donde ***GUI_function*** es una cadena de caracteres (*string*) que contiene el nombre de la correspondiente función interfaz y ***graphics*** es la estructura de datos que contiene los datos gráficos:

graphics= list ([x_o , y_o] , [l , h] , orient , dlg , pin , pout , pcin , pcout, gr_i)

x_o coordenada x del origen del bloque.

y_o coordenada y del origen del bloque.

l ancho del bloque.

h altura del bloque.

orient es un booleano está lanzado o no

dlg es un vectos de cadena de caracteres que contienen los parámetros simbólicos del bloque.

pin vector, pin(i) es el número de enlace conectado al puerto de entrada normal i-ésima y es 0 cuando no están conectados.

pout vector, pout(i) es el número de enlace conectado al puerto de salida normal i-ésima y es 0 cuando no están conectados.

pcin vector, pcin(i) es el número de enlace conectado al puerto de entrada de activación i-ésima y es 0 cuando no están conectados.

pcout vector, pcout(i) es el número de enlace conectado al puerto de salida de activación i-ésima y es 0 cuando no están conectados.

gr_i vector de cadena de caracteres que contiene las instrucciones usadas para dibujar el icono.

La *estructura de datos* que contiene la información de la simulación es ***model*** :

model = list (eqns, #input, #output, #clk_input, #clk_output, state, dstate, rpar, ipar, typ, firing, deps, label,unused)

eqns lista que contiene dos elementos. El primer elemento es una cadena que contiene el nombre de la *función computacional* (fortran, C o función Scilab). El segundo elemento es un entero que especifica el tipo de *función computacional*. El tipo de *función computacional* especifica básicamente la secuencia de llamada, más adelante profundizaremos en esto.

#input vector de tamaño igual al número de puertos de entrada normales del bloque. Cada entrada especifica el tamaño del correspondiente puerto de entrada. Un entero negativo indica que debe ser determinado por el compilador. Si especificamos el mismo entero negativo en mas de un puerto de entrada o salida indica al compilador que esos puertos tienen igual tamaño.

#output vector de tamaño igual al número de puertos de salida normales del bloque. Un entero negativo indica que debe ser determinado por el compilador. Si especificamos el mismo entero negativo en mas de un puerto de entrada o salida indica al compilador que esos puertos tienen igual tamaño.

#clk_input vector de tamaño igual al número de puertos de entrada de activación. Todas las entradas deben ser igual a 1. Scicos no soporta enlaces de activación vectorizados.

#clk_output vector de tamaño igual al número de puertos de salida de activación. Todas las entradas deben ser igual a 1. Scicos no soporta enlaces de activación vectorizados

state vector columna que contiene el estado continuo inicial.

dstate vector columna que contiene el estado discreto inicial.

rpar vector columna de parámetros reales pasados a la correspondiente *función computacional*.

ipar vector columna de parámetros enteros pasados a la correspondiente *función computacional*.

typ cadena que indica el tipo de bloque simple : ' z ' = bloque de cruce por cero, ' l ' = bloque sincronizado y 's' = bloque normal.

firing vector columna de los tiempos de disparo inicial de tamaño igual a el número de puertos de salida de activación de el bloque. Este vector incluye tiempos de disparo de eventos preprogramados (<0 si no hay disparo).

deps [udep timedep]

udep booleano. True si el sistema tiene alimentación directa, o sea que al menos una de las salidas depende explícitamente de una de las entradas.

timedep booleano. True si el bloque es dependiente temporalmente.

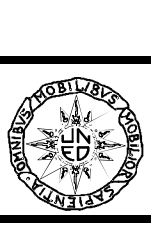
label cadena de caracteres, usada como identificador del bloque. Este campo puede ser seleccionado en el botón **label** del menú **Block**.

5.2. Función Computacional.

La ***función computacional*** evalua las salidas, nuevos estados, estado continuo derivado el vector de tiempos de los eventos de salida dependiendo del tipo de bloque y la forma que sea llamada por el *simulador*.

5.2.a. Comportamiento de Llamada.

El simulador Scicos llama a la *función computacional* para realizar diferentes **tareas** (*task*), utilizando una *variable status* (flag) para especificar que tarea debe ejecutar, como se muestra en la siguiente tabla :



La secuencia de tareas es :

Flag	Task
0	State derivative computation
1	Outputs update
2	States update
3	Output events timing
4	Initialization
5	Ending
6	Re-initialization

Inicialización (Initialization) El simulador llama a la función computacional para conocer el estado del arranque y la salida de inicialización (las entradas no se encuentran disponibles en el momento de arranque). Otras tareas como abrir un fichero, inicialización de la ventana gráfica, etc..., pueden también pueden ser ejecutadas en este punto.

Reinicialización (Re-initialization) El simulador puede llamar al bloque un determinado número de veces para la inicialización. Esta representa otra oportunidad para iniciar los estados y las salidas. Pero esta vez, las salidas se encuentran disponibles.

Actualización de las Salidas (Outputs Update) El simulador llama para obtener los valores de las salidas. De esta manera la función computacional debería evaluar la tarea (4).

Actualización de los Estados (States Update) Uno o más eventos han llegado y el simulador llama a la función computacional para actualizar los estados *x* y *z* de acuerdo con las tareas (2) y (3).

Cálculo del Estado Derivado (State Derivative Computation) El simulador se encuentra en una fase continua ; y se necesita la derivada de x. Esto significa que la función computacional debe evaluar (1).

Tiempo de Salida de Eventos (Output Events Timing) El simulador llama a la función computacional en el tiempo de sus eventos de salida. Para realizar esto la función computacional deberá evaluar (5).

Terminación (Ending) El simulador llama a la función computacional una vez mas al final (es útil para cerrar ficheros, liberar la memoria, etc..).

5.2.b. Tipos de Funciones Computacionales.

En Scicos las *funciones computacionales* pueden ser de diferentes tipos y coexistir en un mismo diagrama de un modelo. Los tipos de funciones existentes vienen especificados en la siguiente tabla:

Function type	Scilab	Fortran	C	Comments
0	yes	yes	yes	Fixed calling sequence
1	no	yes	yes	Varying calling sequence
2	no	no	yes	Fixed calling sequence
3	yes	no	no	Inputs/outputs are Scilab lists

El tipo de *función computacional* utilizada es almacenada en el segundo campo de **eqns** (ver apartado **5.1.b**)

-Función Computacional TIPO 0 El simulador construye un único vector de entrada mediante una pila de todos los vectores de entrada y espera a las salidas apilándolas en un único vector también. Este tipo es soportado por versiones de Scilab anteriores a la 2.4 . La secuencia de llamada es igual que la de **la *función computacional tipo 1*** con una entrada normal y salida normal.

-Función Computacional TIPO 1 La forma más sencilla de explicar este tipo mediante un ejemplo. Los parámetros (I de entrada y O de salida) que utilizan *las funciones tipo 1* son los que muestra la siguiente tabla

I/O	Args.	Description
I	flag	0,1,2,3,4,5 or 6, (see Table 1)
I	nevprt	activation code
I	t	time
O	xdot	derivative of the continuous state
I/O	x	continuous state
I	nx	size of x
I/O	z	discrete state
I	nz	size of z
O	tvec	times of output events (for flag=3)
I	ntvec	number of activation output ports
I	rpar	parameter
I	nrpar	size of rpar
I	ipar	parameter
I	nipar	size of ipar
I	ui	ith input (regular), i=1,2, ...
I	nui	size of ith input
O	yj	jth output (regular), j=1,2, ...
I	nyj	size of jth output

El ejemplo que vamos a tomar es el de **un bloque con dos vectores de entrada normales y cuatro vectores normales de salida :**

Código Fortran :

```
      subroutine myfun(flag,nevprt,t,xd,x,nx,z,nz,tvec,
     &    ntvec,rpar,nrpar,ipar,nipar,u1,nu1,u2,nu2,
     &    y1,ny1,y2,ny2,y3,ny3,y4,ny4)
c
      double precision t,xd(*),x(*),z(*),tvec(*),rpar(*)
      double precision u1(*),u2(*),y1(*),y2(*),y3(*),y4(*)
      integer flag,nevprt,nx,nz,ntvec,nrpar,ipar(*)
      integer nipar,nu1,nu2,ny1,ny2,ny3,ny4
```

Código C Las ***funciones computacionales Tipo 1*** pueden ser también programadas en lenguaje C de la misma forma, pero hay que tener en cuenta que los parámetros deben ser pasados como *punteros*.

-Función Computacional TIPO 2 Este tipo de *función computacional* es especifica para ser programada en *lenguaje C*. Los parámetros que nos encontramos en este tipo de funciones son (I = Entrada y O = Salida)

I/O	Args.	description
I	*flag	0,1,2,3,4,5 or 6, (see Table 1)
I	*nevprt	activation code
I	*t	time
O	xd	derivative of the continuous state (flag= 0)
I/O	x	continuous state
I	*nx	size of x
I/O	z	discrete state
I	*nz	size of z
O	tvec	times of output events (flag=3)
I	*ntvec	number of activation output ports
I	rpar	parameter
I	*nrpar	size of rpar
I	ipar	parameter
I	*nipar	size of ipar
I	inptr	inptr[i] is pointer to beginning of ith input
I	insz	insz[i] is the size of the ith input
I	*nin	number of input ports
I	outptr	outptr[j] is pointer to beginning of jth output
I	outsz	outsz[j] is the size of the jth output
I	*nout	number of output ports

Un resumen de cómo programar esta función es el siguiente :

```
#include "<SCIDIR>/routines/machine.h"
void selector(flag,nevprt,t,xd,x,nx,z,nz,tvec,ntvec,
   rpar,nrpar,ipar,nipar,inptr,insz,nin,outptr,outsz,nout)

integer *flag,*nevprt,*nx,*nz,*ntvec,*nrpar;

 integer ipar[],*nipar,insz[],*nin,outsz[],*nout;

 double x[],xd[],z[],tvec[],rpar[];
 double *inptr[],*outptr[],*t;
```

Ejemplo : Se muetra **como ejemplo** el código C del ***bloque Selector*** . Se asume que en los eventos pueden llegar a los puertos de entrada de eventos del bloque a un mismo tiempo.

```
#include "../machine.h"
void selector(flag,nevprt,t,xd,x,nx,z,nz,tvec,ntvec,
  rpar,nrpar,ipar,nipar,inptr,insz,nin,outptr,outsz,nout)

integer *flag,*nevprt,*nx,*nz,*ntvec,*nrpar;
integer ipar[],*nipar,insz[],*nin,outsz[],*nout;

double x[],xd[],z[],tvec[],rpar[];

double *inptr[],*outptr[],*t;
{
    int k;
    double *y;
    double *u;
    int nev,ic;
    ic=z[0];
    if ((*flag)==2) {
    /* store index of input event port fired */

       ic=-1;
       nev=*nevprt;
       while (nev>=1) {
           ic=ic+1;
           nev=nev/2;
       }
       z[0]=ic;
   }
   else {
   /* copy selected input on the output */

            if (*nin>1) {
                y=(double *)outptr[0];
                u=(double *)inptr[ic];
                for (k=0;k<outsz[0];k++)
                    *(y++)=*(u++);
            }
            else {
                y=(double *)outptr[ic];
                u=(double *)inptr[0];
                for (k=0;k<outsz[0];k++)
                    *(y++)=*(u++);
            }
        }
    }
```

-Función Computacional TIPO 3 Este tipo de *función computacional* es específica para ser programada en *Scilab*. La secuencia de llamada es la siguiente :

```
[y,x,z,tvec,xd]=test(flag,nevprt,t,x,z,rpar,ipar,u)
```

La **descripción de sus parámetros** (I = Entrada O = Salida) es la siguiente :

I/O	Args.	description
I	flag	0,1,2,3,4,5 or 6 (see Table 1)
I	nevprt	activation code (scalar)
I	t	time (scalar)
I	x	continuous state (vector)
I	z	discrete state (vector)
I	rpar	parameter (any type of scilabtt variable)
I	ipar	parameter (vector)
I	u	u(i) is the vector of i th regular input (list)
O	y	y(j) is the vector of j th regular output (list)
O	x	new x if flag=2, 4, 5 or 6
O	z	new z if flag=2, 4, 5 or 6
O	xd	derivative of x if flag= 0 (vector), [] otherwise
O	tvec	times of output events if flag=3 (vector), [] otherwise

Ejemplo : La siguiente *función computacional* asociada a un bloque que cada cierto tiempo recibe un evento y nos va a mostrar en una ventana Scilab los eventos recibidos hasta el presente y los valores de sus dos entradas.

```
function [y,x,z,tvec,xd]=test(flag,nevprt,t,x,z,rpar,ipar,u)
y=list();tvec=[];xd=[]
if flag==4 then
    z=0
elseif flag==2 then

    z=z+1
    write(%io(2),'Number of calls:'+string(z))
    [u1,u2]=u(1:2)
    write(%io(2),'first input');disp(u1)
    write(%io(2),'second input');disp(u2)
end
```

Ejemplo : La ventaja de programar las salidas y las entradas como *listas* es que el número de entradas y de salidas no necesita ser especificado explícitamente. En este **ejemplo**, la salida es el elemento obtenido de todos los vectores entrada, independientemente del número de entradas.

```
function [y,x,z,tvec,xd]=elemprod(flag,nevprt,t,x,z,rpar,ipar,u)
tvec=[];xd=[]
y=u(1)
for i=2:length(u)
   y=y.*u(i)
end
y=list(y)
```

6. CONCLUSIÓN.

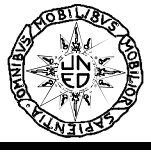
Este trabajo quiere dar sólo una idea general descriptiva y de uso de Scicos. Más información puede ser encontrada en la ayuda de Scilab dentro de la librería de Scicos, las demos de Scicos en Scilab son una interesante fuente de información y las referencias que aparecen en el siguiente apartado.

7. REFERENCIAS.

1. **Manual de Introducción al Tratamiento de Señales con SCILAB para usuarios de MATLAB.** Autor : Doctorando Bernardo A. Delicado (Departamento de Informática y Automática de la UNED). ftp://ftp.inria.fr/INRIA/Projects/Meta2/Scilab/contrib/delicado/

2. **Introduction To Scilab (User's Guide).** Autor : Scilab Group (Institute National de Recherche en Informatique et en Automatique (INRIA) France).

3. **SCICOS A Dynamic System Builder and Simulator (User's Guide) .** Autores : R. Nikoukhah y S. Steer (Institute National de Recherche en Informatique et en Automatique (INRIA) France).


OMNIBVS MOBILIBVS SAPIENTIA
UNED