

# Scicos: a dynamic systems modeler and simulator

Ramine Nikoukhah

INRIA-Rocquencourt

Domaine de Voluceau, 78153 Le Chesnay Cedex, France

[Ramine.nikoukhah@inria.fr](mailto:Ramine.nikoukhah@inria.fr)

**Keyword:** Modeling, simulation, hybrid systems

## Abstract

The underlying formalism of the dynamic system modeler and simulator Scicos is presented. Using simple examples, it is shown how hybrid (continuous/discrete time) models can be constructed in this formalism. It is also shown that this formalism is closely related to synchronous languages.

## Introduction

Scicos is an open-source Scilab toolbox included in the Scilab Package providing many functionalities available in *Simulink* and *MATRIXx*. Based on an open formalism motivated by synchronous languages extended to continuous time dynamics, Scicos can be used to model and simulate hybrid dynamical systems. This paper presents some aspects of the Scicos formalism by studying simple examples.

## Scicos environment

Scicos provides a complete environment for construction of models, simulation and code generation. It includes: a **graphical Editor**. Scicos provides a hierarchical graphical editor for the construction of models of dynamical systems using block diagrams. A number of predefined blocks are also provided in various palettes. New blocks can be defined by the user in C, Fortran or Scilab.

It includes a **compiler**: Scicos compiler uses the model description, usually compiled by the Scicos editor, to construct scheduling tables which can then be used by the simulator and the code generation function.

It includes a **simulator**: Scicos simulator uses the scheduling tables and other information provided by the compiler to run simulations. The simulator is of hybrid nature in that it has to deal with discrete and continuous time systems, and events. For the continuous time part, it uses the ODE solver *LSODAR* or the DAE solver *DASKR* depending on the nature of the continuous time system considered. And it includes a **code generator**: Scicos can generate C code for realizing the behavior of some

subsystems. These subsystems should not include continuous time components.

## Scicos and Scilab

Scicos is a Scilab toolbox and runs in the Scilab environment. Having access to Scilab functions when designing simulation models is of great importance: Scicos user often needs to use Scilab functions such as those dedicated to filter design for signal processing or controller design in the construction of simulation models. Scilab programming language can be used for batch processing of multiple simulation tasks, and more generally, models designed by Scicos can be used as functions in Scilab. Scilab graphical facilities can be used for post processing simulation results.

## Scicos Formalism

Scicos is a tool for designing reactive systems. Scicos models are designed using a block diagram editor, but an underlying language exists providing a well defined formalism. This formalism is very simple because it deals exclusively with the reactive part of the design; it does not provide a complete programming language. The blocks are considered as atoms in Scicos; Scicos simulator considers them almost as black boxes. It knows some of their properties but not the underlying code. The code realizing the behavior of a block (called the simulation function) can be written in C, Fortran or Scilab. In Scicos formalism, the execution of simulation functions are considered instantaneous so Scicos can be considered a Synchronous language or more specifically an extension of it to handle continuous time systems. The existence of a unique universal time is assumed in the formalism.

## Scicos Block

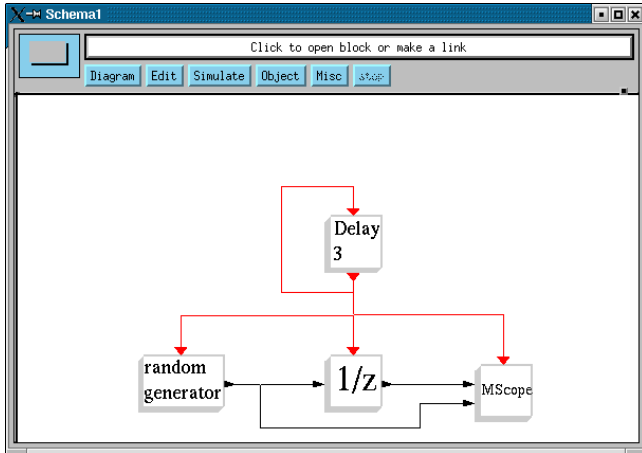
A Scicos block can have two types of inputs and outputs:

- regular inputs (usually placed on the sides)
- regular outputs (also on the sides)
- activation inputs (usually on top)
- activation outputs (usually at the bottom)

Regular inputs and outputs are used to communicate data from block to block through regular links. Activation inputs and outputs are connected by activation links which transmit control information (activation).

## Example

Consider the following example which displays a random sequence and its delayed version on a multi-scope.

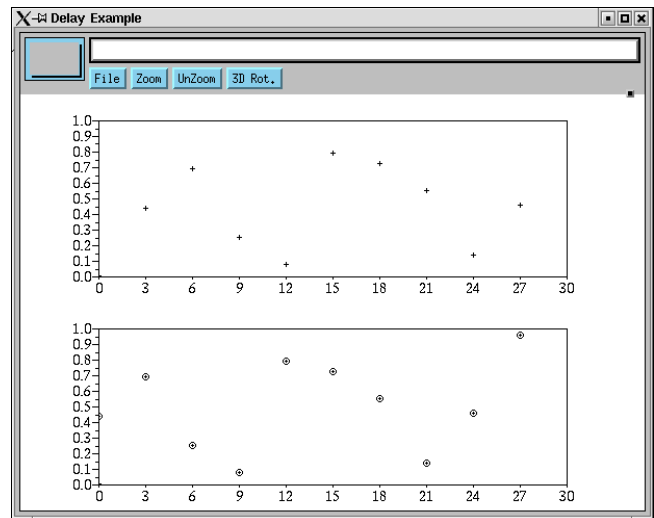


There are four blocks here:

- **random generator**: at each activation, a new random number is placed on its sole regular output
- **1/z**: just a memory, at activation, the internal state is copied to the regular output and then the input is copied into state
- **MScope**: Scilab graphics window emulating a multi-display scope
- **Delay**: when activated, this blocks programs an activation output for some later time (here 3). An activation event is originally programmed at time 0 on its activation output port.

The diagram contains only one source of activation signal (the output of Delay). So all the blocks are synchronized and are activated simultaneously. This includes the MScope which is just a simple block and its activation implies reading the values on the inputs, placing them in a buffer and displaying.

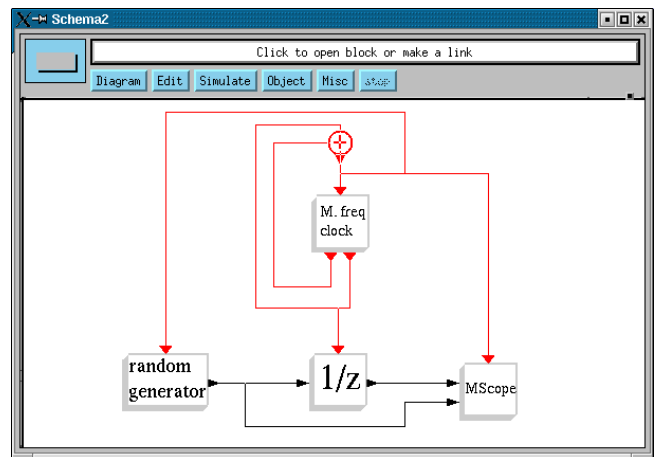
## Simulation result



In the second plot, the delay is 3 units of time.

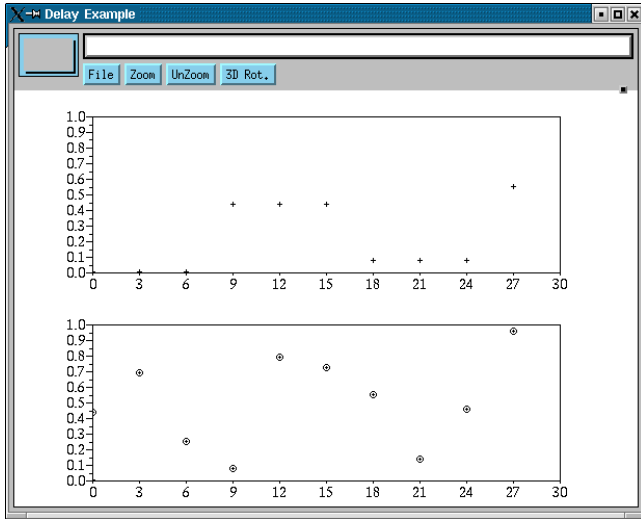
## Multifrequency activation

An activation clock is simply an event delay with output fed back to its input. This generates a sequence of regularly spaced in time activation events. A multi-frequency activation clock can be similarly constructed. For that, we use an event delay block with two output activation ports. The delayed activation is programmed once on one port then n-1 times on the other. n is a block parameter. See the following example:



The **addition** on activation signals represents the union of activation times. The **M.-freq-clock** block here has a delay of three and the parameter n is also set to 3. Thus this block, when activated, programs an event on its left output port for, for 3 units of time later. It does it again when reactivated. But the next time, it programs it on the right output port and the whole process starts over again (this behavior is achieved using an internal state which starts from 1 and it is incremented at each activation until it hits n when it is set back to zero). By feeding back all the delayed events into the input activation port of this block, we end up with a multi-frequency activation clock: the sum (union) of the two delayed activation signals is a regularly spaced in

time sequence of events (with period 3), and the right output of the block generates also a regularly spaced in time sequence of events but with period equal to 9.



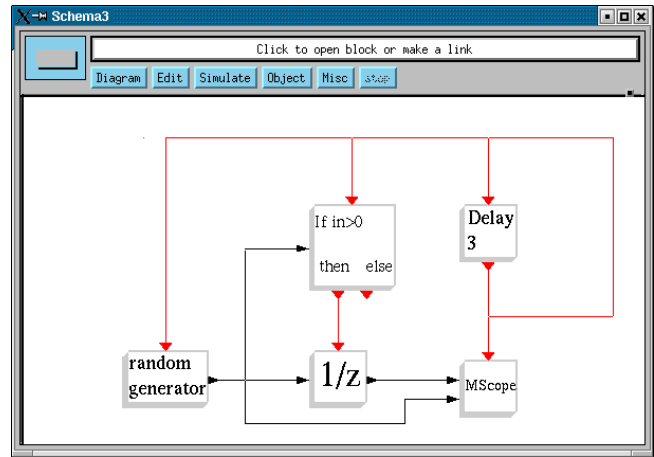
This example does not represent a synchronous system because there are two sources of activation signals. Sources of activation signals are considered asynchronous and an asynchronous diagram present in general non deterministic behavior. In this particular example however there is no non-determinism possible as long as the delay in the M-freq-clock block is strictly positive. Note that the two activation signals used for activating other blocks are synchronous. In particular the signal activating the  $1/z$  block is a subsample of the signal activating the random generator and the MScope blocks.

## Subsampling

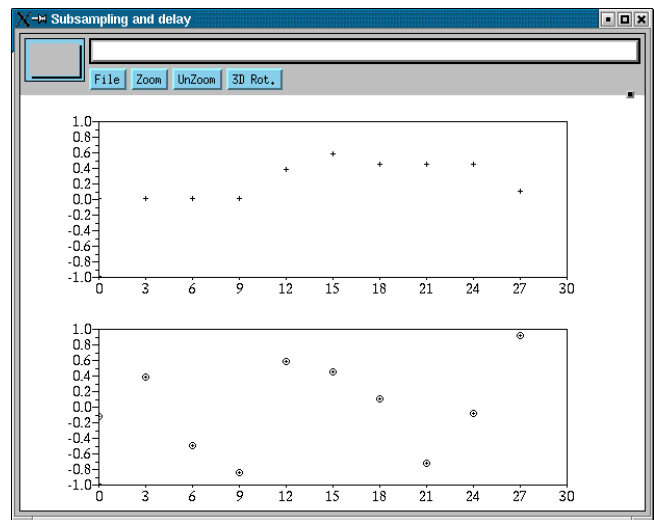
There are two conditional blocks in Scicos which implement conditioning. Even though they are presented and used as blocks in the editor, strictly speaking they are not Scicos blocks. They have no corresponding simulation functions and they are the only « blocks » that generate output activation signals synchronized with their input activation signals. The activation outputs of these block do not constitute independent and asynchronous sources of activation.

The two blocks should be compared to the **If-Then-Else** and **Switch** statements in C, but of course the context here is very different.

The multi-frequency behavior seen in the previous example can be achieved in a purely synchronous context using subsampling. In the following example, this point is illustrated. To make the example more interesting, the subsampling is not done based on a fixed frequency but conditioned. In particular, the activation for the  $1/z$  block is generated only when the output of the random generator is positive.



The simulation result given below illustrates exactly what the system does.



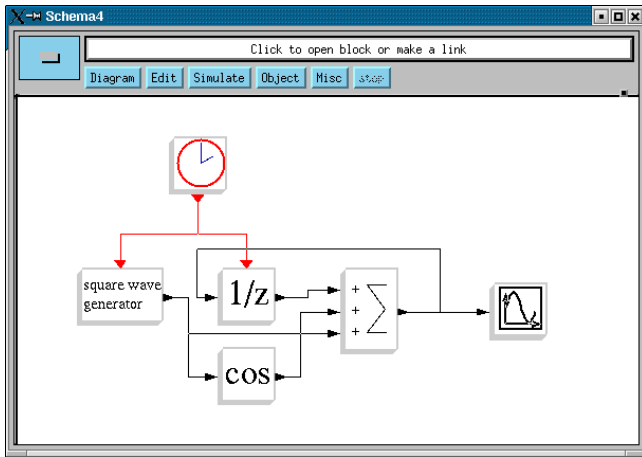
This diagram is synchronous because it contains only one source of activation signal. Note that the If-Then-Else « block » is not a block as far as Scicos is concerned. It is represented as a block in the editor for convenience. The two activation outputs of this block are mutually exclusive and their sum (union of activation times) equals its input activation. In a sense, this block redirects the activation it receives in one or the other direction; it does not consume it as a Scicos block would.

## Event Driven vs Data Flow

Strictly speaking Scicos provides an event driven environment. This means the activation of each block is explicitly determined by an activation signal. However, through the inheritance mechanism, Scicos provides to some extent a data flow behavior as well.

## Inheritance

Consider the following example.

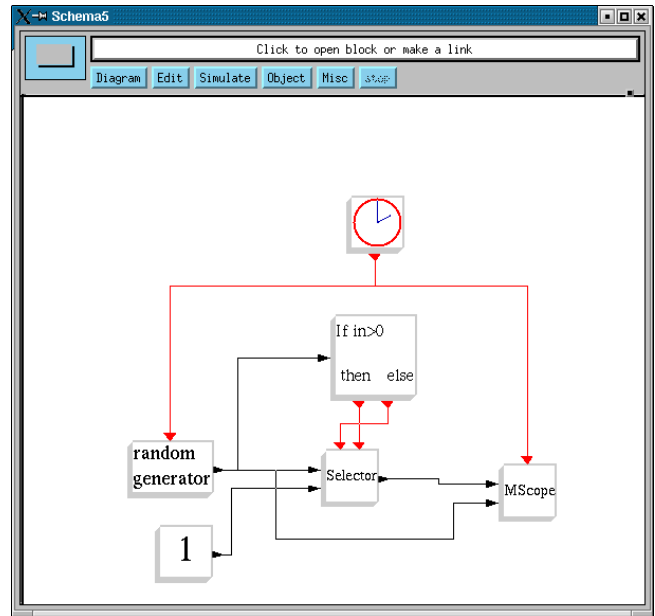


Here that unlike previous examples, we have blocks which do not have activation input ports. So how are they activated? The answer is: by inheritance. A block with no activation input port (and not always active, we shall see that later) inherits its activation through its regular inputs. The activation associated with a regular signal is the activation times of the block which has generated the signal. In this example we have only one source of activation, the Event Clock (Event Clock is not a block but a super block which includes a Delay block with a feedback as seen before). The Cos block inherits its activation from its regular input signal generated by the Square Wave Generator block. This latter is activated by the activation signal from the Event Clock, so due to the inheritance mechanism, we have that the Cos block is also activated by this activation signal. In fact we would have exactly the same model if the Cos block had an activation input port and the output of the Event Clock was connected to it.

The Summation and Scope blocks are also activated by inheritance. In fact, the  $1/z$  block would also function the same way if it did not have an activation input port. But if it does and it is not connected, the block remains inactive (no inheritance).

## Inheritance and multifrequency

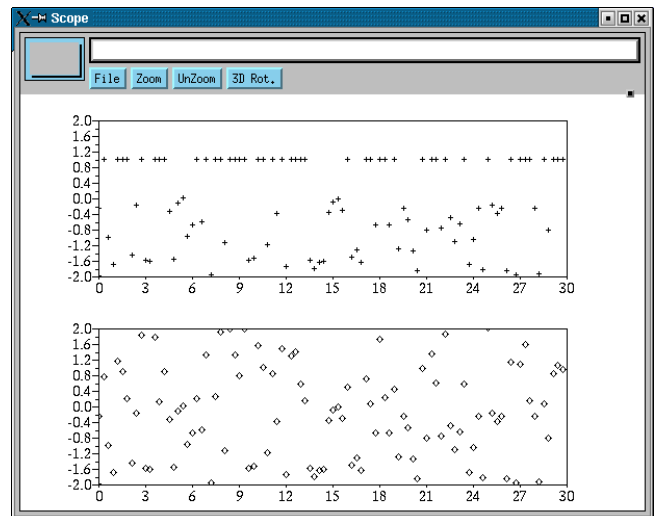
The inheritance mechanism is pretty simple to understand when a single activation exists in the model. With the presence of subsampling or in asynchronous cases, the inheritance still works following specific rules. Let us start by noting that blocks in Scicos may have more than one input activation ports.



The block **Selector** has two input activation ports. The block is activated by both activation signals. There are three activation scenarios possible for a block with two activation input ports: block is activated through its first activation input port, block is activated through its second activation input port, block is activated through both ports.

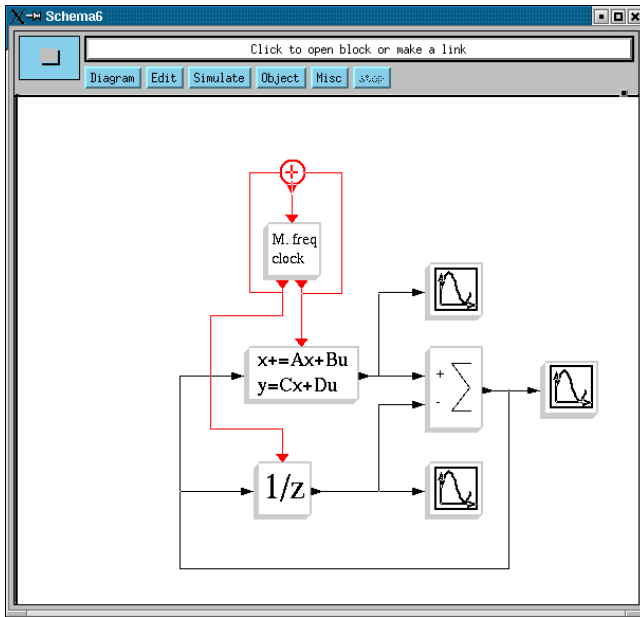
The block knows by which way it has been activated. **Selector** uses this information to place on its output the first or the second input depending on the port through which it has been activated. In this case, the last scenario does not occur because activation signals out of the If-Then-Else block are mutually exclusive.

It is now easy to see what the diagram does. The output of the **Selector** is the random signal generated by **Random Generator** block as long as the signal is negative or zero. If it is positive, the output of the **Selector** is 1. Here is the simulation result.



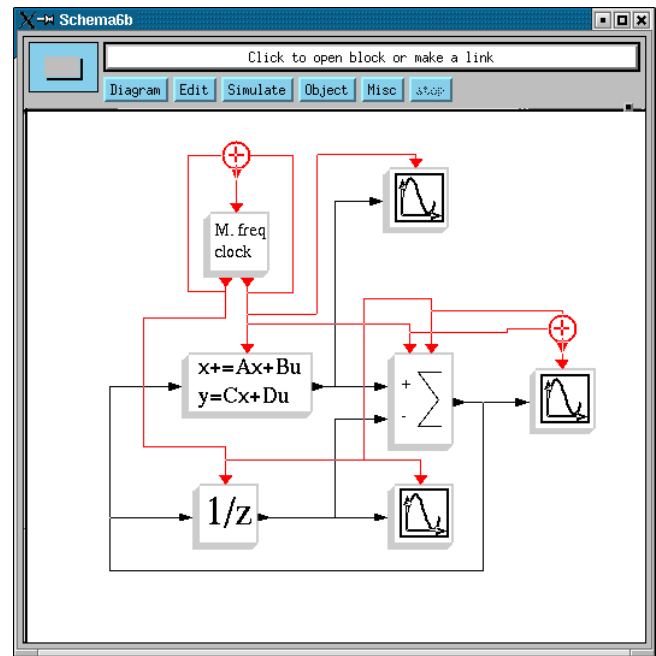
Note on this example the presence of the block **Constant** (=1). This block is not activated and it clearly cannot inherit any activation (absence of input); it is not declared **always active** either (more on the property of **always active**, later). Such blocks can only generate constant outputs because they are «activated» only once at the initial time (and possible re-initialization in a special case discussed later).

Going back to the problem of inheritance consider the following example.



The parameters of the **M.-freq-clock** block are chosen such that every 3 units of time it is activated once, and its activation goes once to the block realizing the discrete linear system and twice to **1/z** block

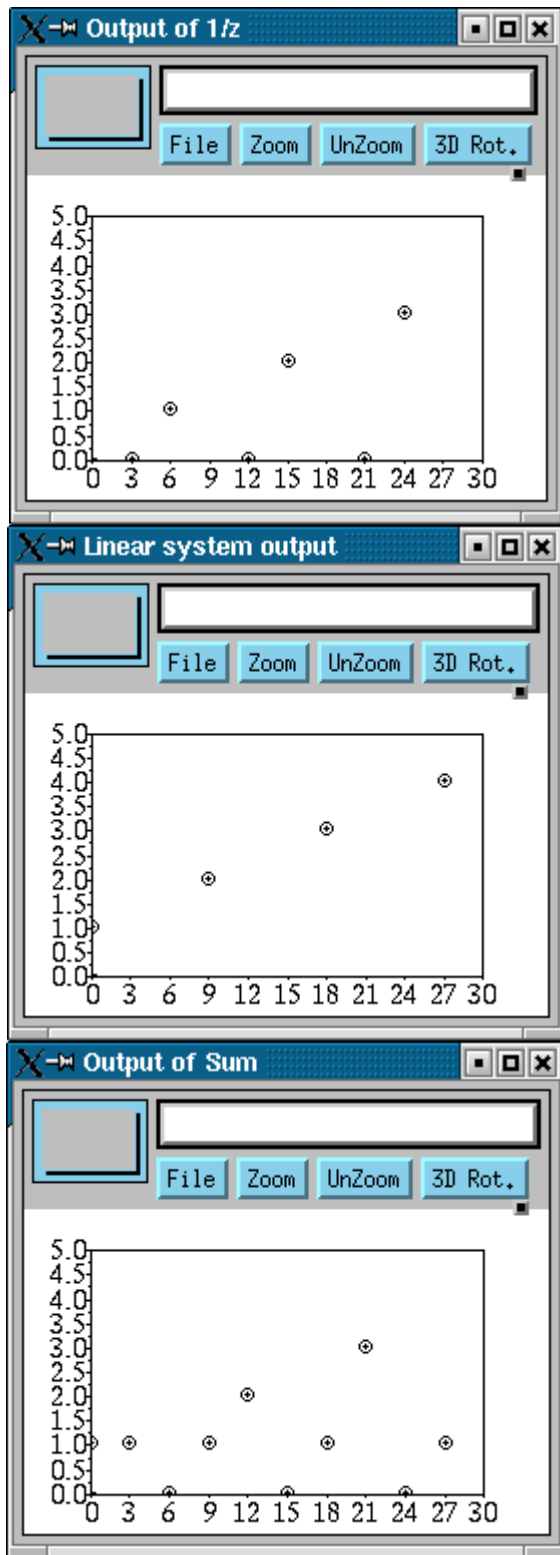
In this example the **Summation** and the **Scope** blocks all work by inheritance. The **Scope** blocks on top and at the bottom inherit clearly from the linear system and **1/z** blocks. In such simple cases, this means that an activation input port is added to the block which inherits (in this case **Scope**) and its is connected to the same activation source which is activating the block generating its regular input (linear system for the first **Scope** and **1/z** for the second). The case of the **Summation** block is a bit more complicated; it inherits from two input signals. In this case, the inheritance mechanism creates two input activation ports for the block and connect them according to the sources of the regular inputs. The way Scicos handles the inheritance in this case can be seen in the following diagram which is completely equivalent to the original diagram but where all the activations are explicit.



The **Summation** block does not use the information available to him as to through which port it has been activated (it could have). Every time it is activated, it simply performs a sum (actually here a subtraction). It is activated every time the linear system block or **1/z** is activated—its activation corresponds to the sum of the activations of these two blocks. This of course makes sense because the times where the sum operation should be performed corresponds to the union of times when its inputs may change. This in fact is the underlying justification for inheritance.

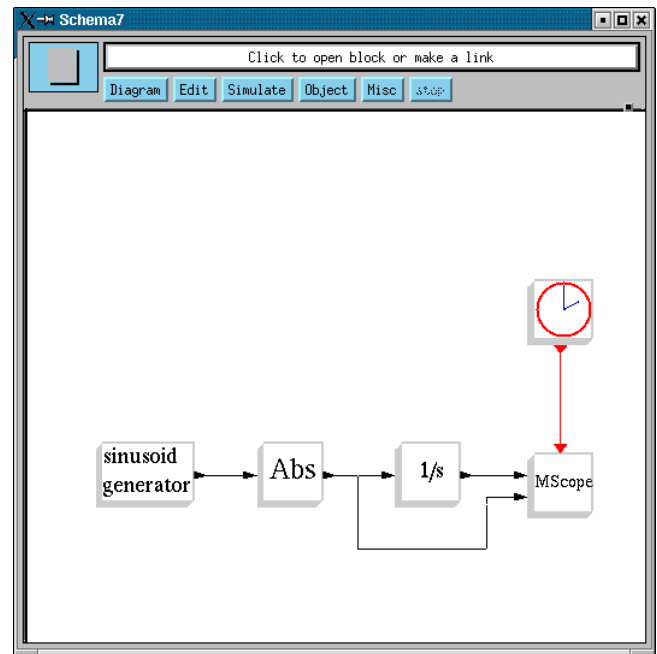
The last **Scope** inherits from the **Summation**. It has only one regular input port, so inheritance mechanism creates a single input activation port. This port receives the activations of the **Summation** block, i.e., the sum of the activations received on its two input activation ports.

The simulation result illustrates how the scopes are activated due to the inheritance mechanism.



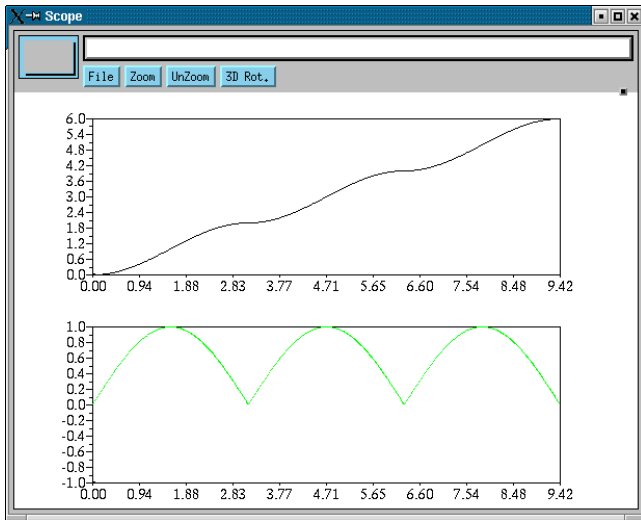
A block can be declared always active. This means in particular that the block is a continuous-time block, i.e., its outputs and state can vary continuously. Normally, always active property should be designated by an activation signal received on an activation input port if we are to be consistent with Scicos formalism, however, to avoid useless complexity at the level of the editor, this property is simply specified as a block property.

There are a number of blocks in Scicos palettes which are always active. These blocks can be used with other blocks to construct hybrid diagrams.



In this example the Sinusoid Generator and the integrator block 1/s have the always active property (not visible on the diagram, it is an internal property). The Abs block inherits this property. If 1/s did not have the always active property, it would have inherited it and the result would have been the same. In most cases, an integrator without the always active property would work except in special situations for example when it is placed in front of a constant block. The Event Clock here is simply used to fix the pace at which the MScope displays its input signals. The result is:

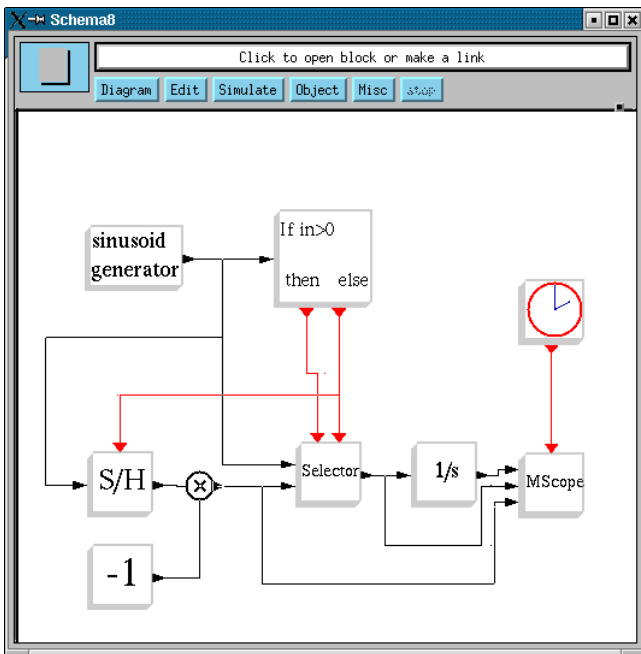
## Continuous time



Note that the output of the continuous time block 1/s is connected directly to the input of the event driven block MScope. The value taken by the latter is simply the sampled value of the continuous time signal.

## Continuous time and subsampling

Subsampling also works with continuous time signals as well. Consider the following system:

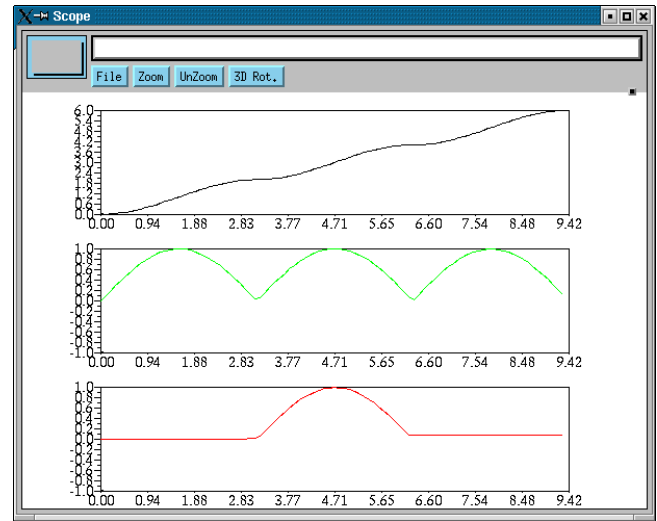


In this example, we again compute the integral of the absolute value of  $\sin(t)$ . This is done by selecting  $\sin(t)$  or  $-\sin(t)$  depending on the sign of  $\sin(t)$ .

The block S/H (Sample and hold) is not necessary here; the simulation result would be the same without it. We have used it to illustrate an important point in relation with inheritance. Without it, the block Multiply would be active

all the time (inheriting from Sinusoid Generator). But now, it inherits from the Else port if the If-Then-Else block, which means that it is only active when  $\sin(t)$  is negative. Of course in this simple example, that is not so important. But if in place of the block Multiply we had a large number of complex blocks, the economy would be significant.

The simulation result shows the « inactivity » of Multiply when the result is not needed.

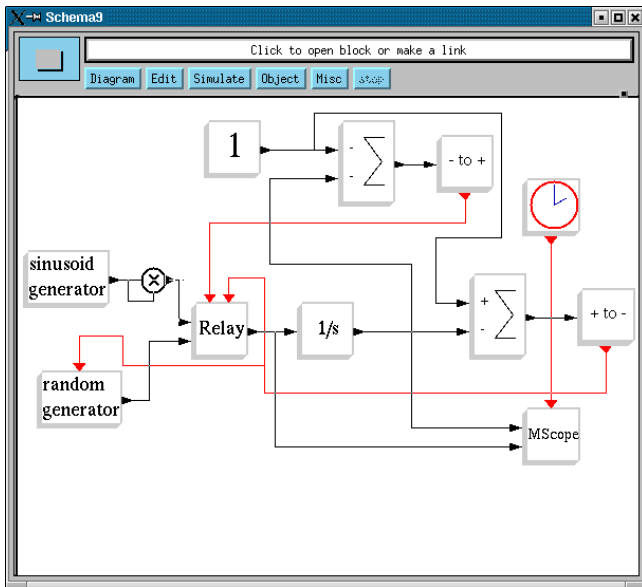


## Continuous discrete time interaction

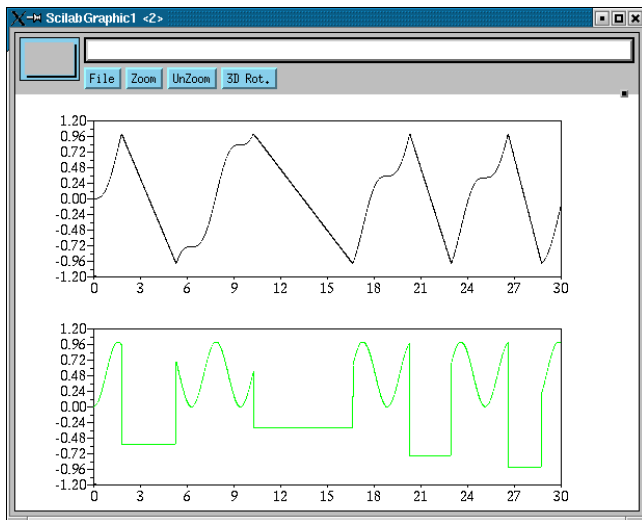
Continuous time operations and discrete-time event dependent operations can interact in different ways. Continuous and discrete time signals can be inputs to the same block. In fact fundamentally, there is no difference between a discrete time signal and a continuous time signal. In fact a Scicos signal can have discrete property over a period of time and later continuous property. This means that in **Scicos** we can perform operations (such as addition of) continuous and discrete time signals. Continuous time signals can **generate events** through **zero-crossing blocks**. Events can create jumps in continuous time states.

The following example shows some of the above points.

R. Nikoukhah, S. Steer, "Conditioning and hybrid system formalism," ADPM, Dortmund, Germany, Sept. 2000.



The blocks **-to+** and **+to-** are **zero-crossing blocks** which generate events when their continuous time input signals cross zero respectively with a positive and a negative slope. These events are used here in the **Relay** block to change the input of the integrator block **1/s**. The simulation result is given below.



## Conclusion

Various aspects of Scicos formalism has been presented. It is shown in particular how activation and inheritance are used to obtain an environment for modeling hybrid dynamical systems in a precise and simple manner.

## Bibliography

R. Nikoukhah, S. Steer, "Scicos: a hybrid system formalism," ESS'99, Erlangen, Germany, Oct. 1999.