

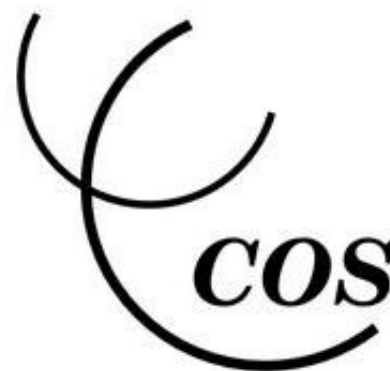
Introducing ScicosLab and Kepler



Modeling, simulation and and controller design using ScicosLab and Kepler

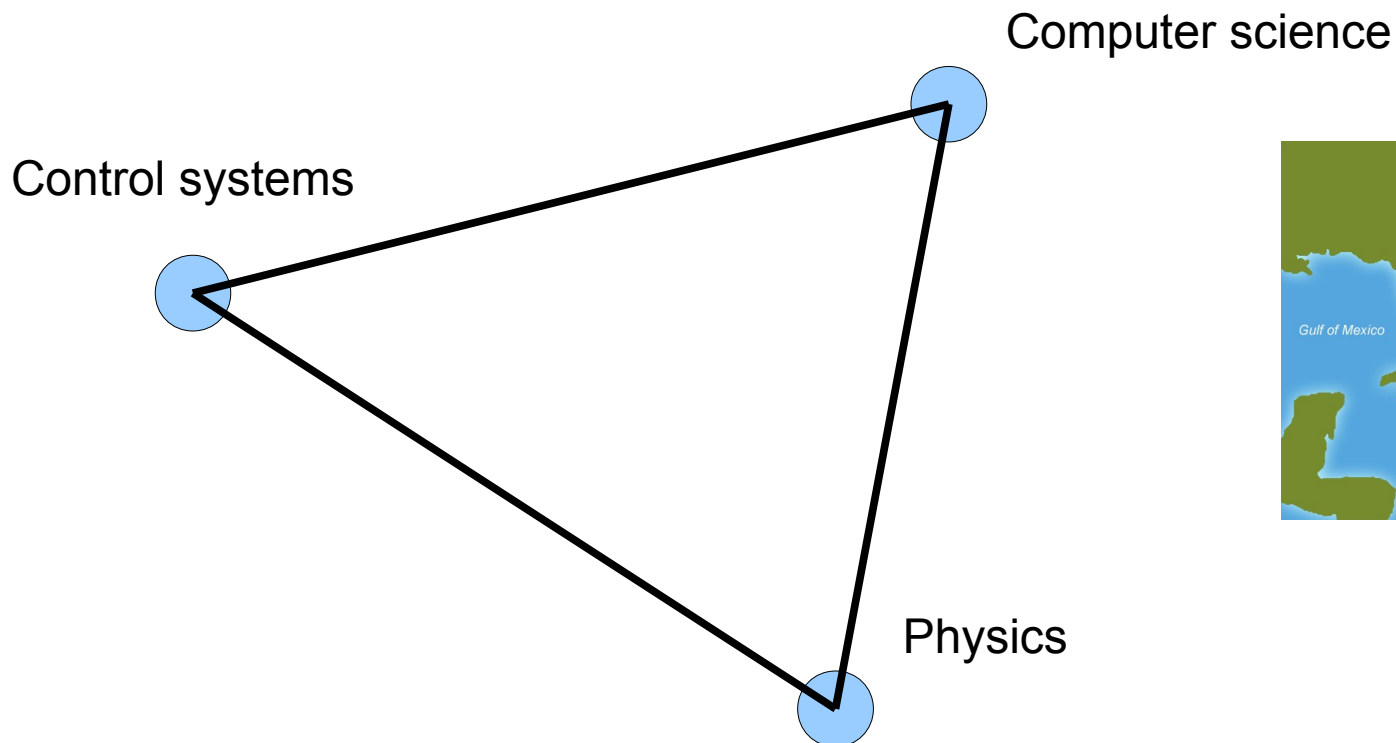


www.scicoslab.org

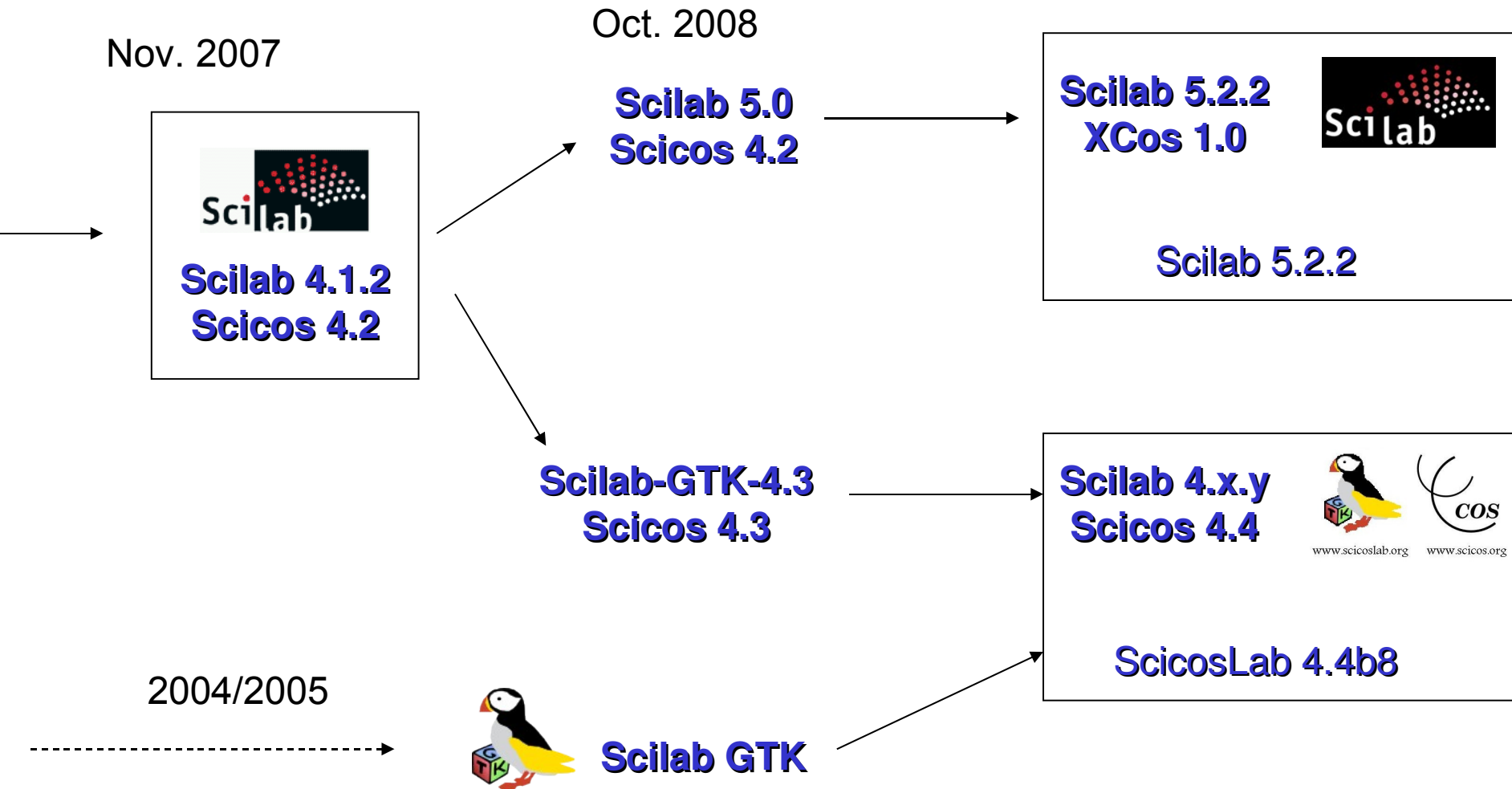


www.scicos.org

Digital control systems design and simulation: the Bermuda Triangle for engineers



Scilab / Scicos / Scilab-GTK / ScicosLab timeline



What ScicosLab is?

- An interpreted language (“Scilab Language”, very similar but not equal to Matlab)
- Full support for matrix computation (BLAS, LAPACK, single and multi cores support using ACML now, GPU support in the near future using OpenCL).
- Basic programming (just like Matlab)
- GUI development (using TCL/TK and UICONTROL)
- Linear algebra
- Polynomial calculation
- Control systems modelling and design (continuous, discrete and hybrid systems)
- Robust control toolbox
- Optimization and simulation (build-in solvers)
- Signal processing (filters design, etc.)
- ARMA modelling and simulation
- Basic statistics
- Basic identification
- PVM support
- TCL/TK and Java support
- Max-plus algebra toolbox

What you can do with ScicosLab?

- Interact with the command line
- Write a program, one line at time, using the internal or with an external editor
- Run the program in an user friendly interpreted environment (easy debug)
- Use the rich embedded library.
- You can develop your ScicosLab functions using C, Fortran, Java, etc.

Produce nice graphics diagrams.

What Scicos is?

Scicos is a graphical dynamical system modeler and simulator developed inside the METALAU project at INRIA, Paris Rocquencourt centre.

Scicos is a **graphics**, object oriented tool where the user create block diagrams to model and simulate the dynamics (**time domain**) of **hybrid** dynamical systems and compile models into executable code (code generation for simulation and embedded applications).

Scicos is used for signal processing, systems control, queuing systems, and to study physical and biological systems.

New extensions allow generation of component based modeling of electrical and hydraulic circuits using the **Modelica** language.

What you can do with Scicos?

- Graphically model, compile, and simulate dynamical systems
- Combine continuous and discrete-time behaviours in the same model
- Select model elements from “Palettes” of standard blocks
- Program new blocks in C, Fortran, or Scilab Language
- Run simulations in batch mode from ScicosLab command line
- Generate C code from Scicos model using the built in Code Generator
- Generate C/C++ libraries ready to be integrated in Kepler using FC2K
- Run simulations in real time with and real devices using Scicos-HIL
- Generate hard real-time control executables with Scicos-RTAI, Scicos-FLEX and **Scicos-ITM** code generators.
- Use implicit blocks developed in the Modelica language.
- Discover new Scicos capability using additional toolboxes like RTSS, Scicos-HDL, Coselica, etc.

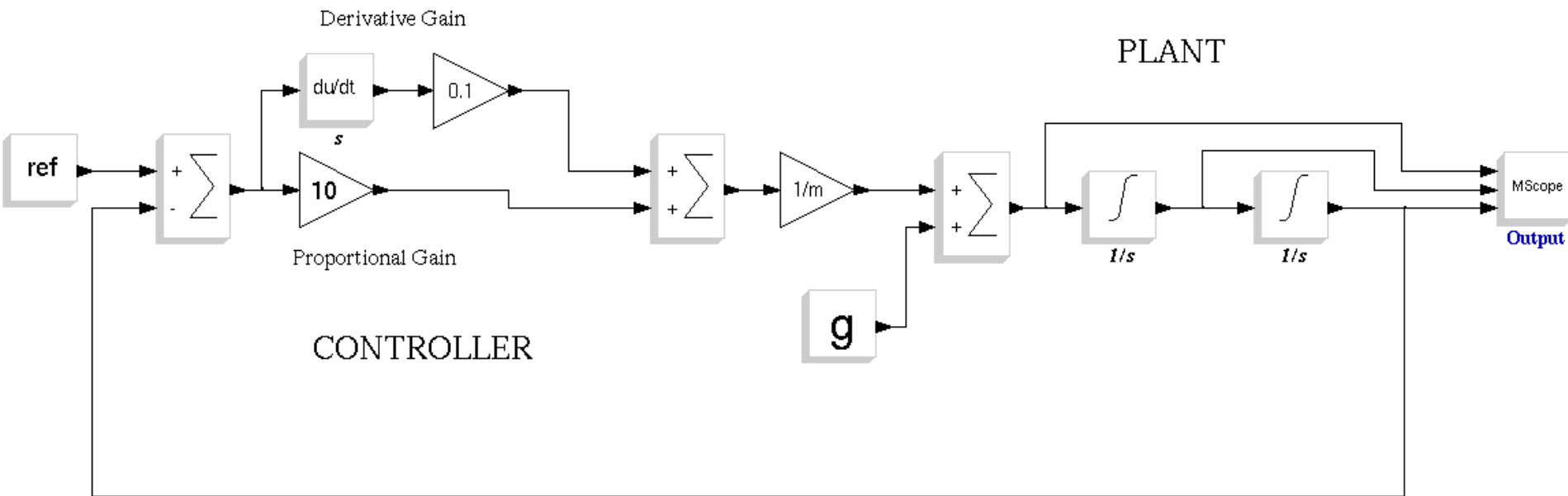
Modeling of complex dynamical systems

ScicosLab and Scicos are designed to handle explicit (ODE) or implicit (IDA) differential equation problem (in the continuous domain) and difference equation (in the discrete domain).

ScicosLab and Scicos can handle also partial differential equation (PDE) and finite element problem, but the user must develop some code to implement features like meshing (not built in ScicosLab).

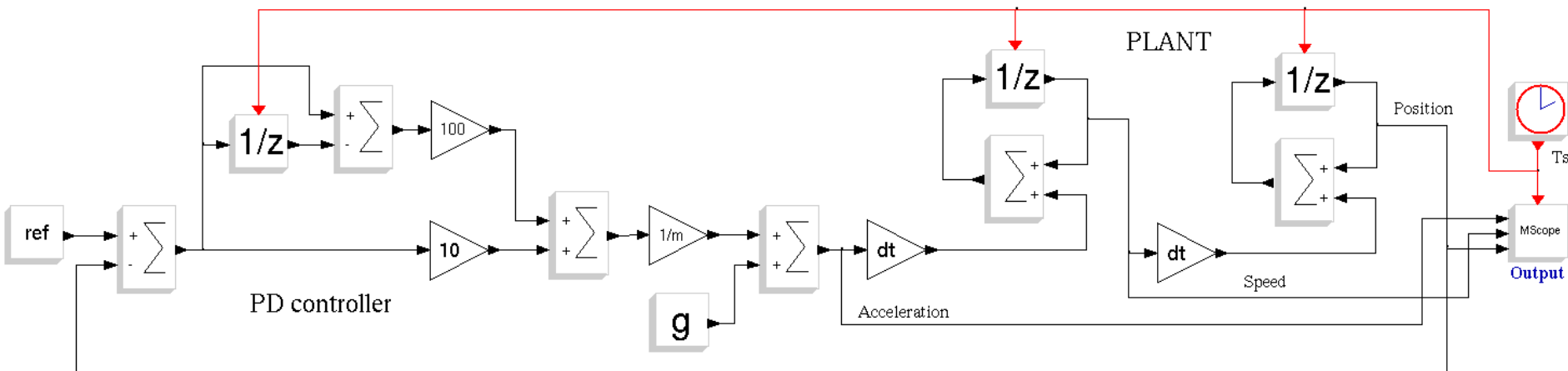
Continuous systems

Linear controller for a floating apple.



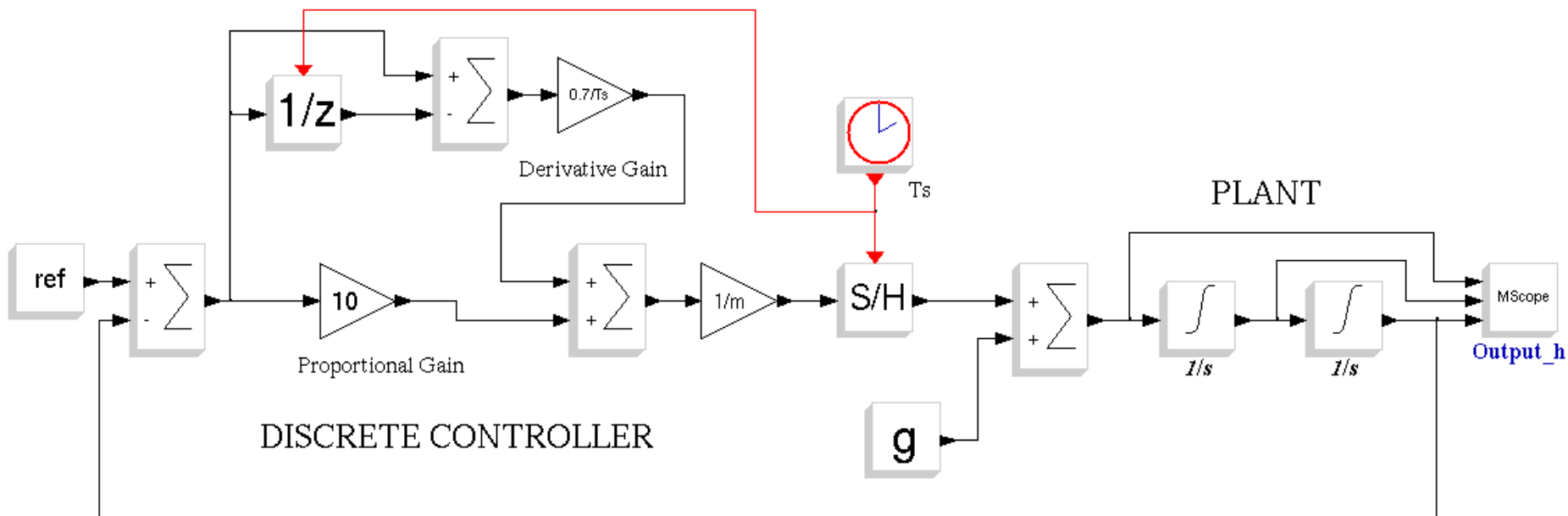
Discrete systems

Same diagram, but realized with (time) discrete blocks.



Hybrid systems

Welcome to the Real World: continuous system, discrete controller.

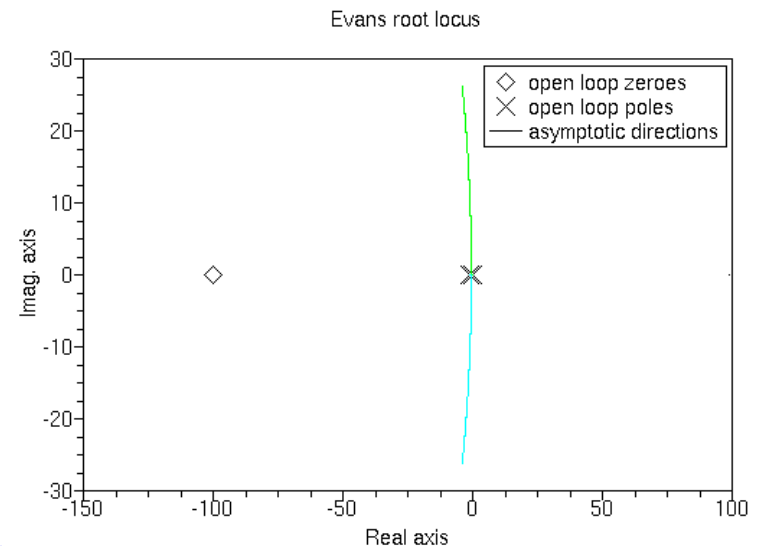
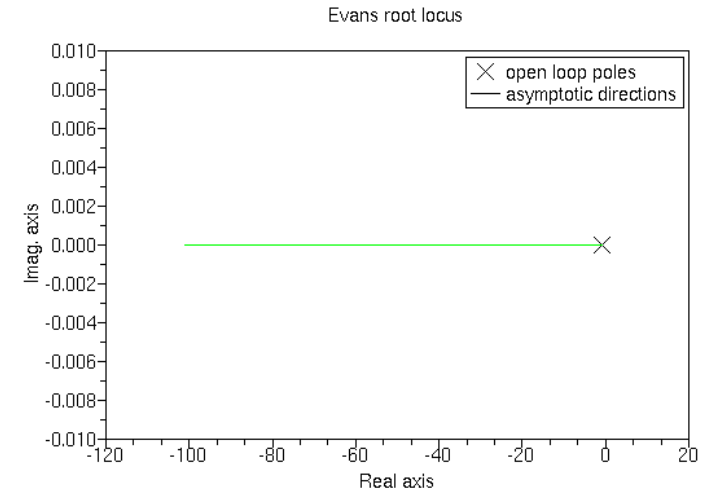


Root locus and Bode plots with ScicosLab

```

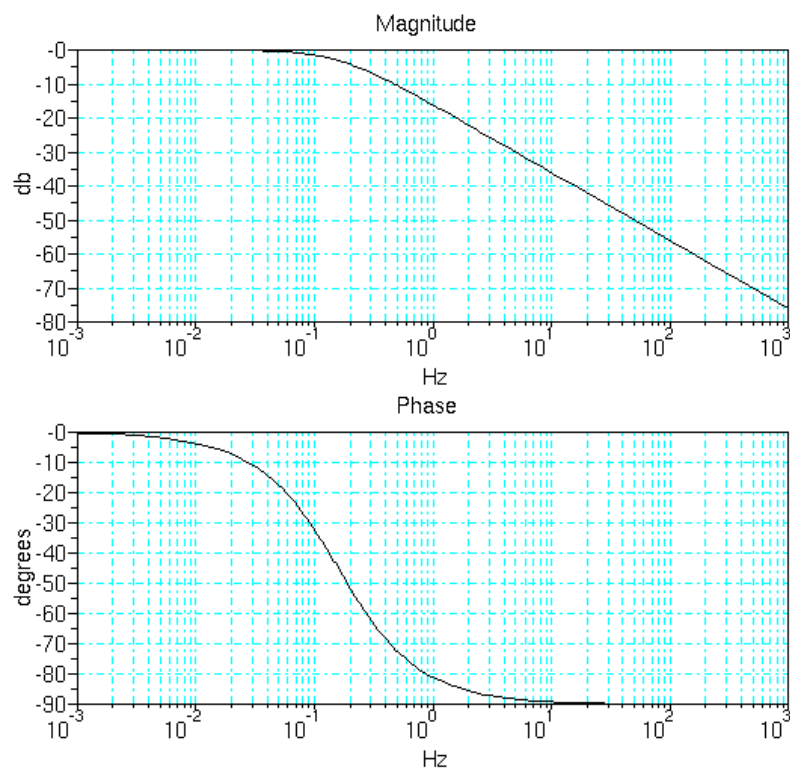
1 /*** Plant parameters
2 R = 1.0; L = 1.0;
3 /*** PI controller parameters
4 Kp = 1; Ki = 100 ;
5 /*** sys plant build
6 N_plant = poly([1], "s", "coeff");
7 D_plant = poly([R L], "s", "coeff");
8 sys_ol = syslin("c", N_plant, D_plant);
9 /*** plant root locus
10 Kmax = 100;
11 scf(0); evans(sys_ol, Kmax);
12 disp("With PI controller"); pause
13 /*** sys PID build
14 N_pid = poly([Ki Kp], "s", "coeff");
15 D_pid = poly([0 1], "s", "coeff");
16 /*** controller + plant
17 N = N_pid * N_plant;
18 D = D_pid * D_plant;
19 /*** full system
20 sys_cl = syslin("c", N, D);
21 scf(1); evans(sys_cl);
22 /*** plant Bode diagram
23 scf(2); bode(sys_ol);
24 /*** controller + plant Bode diagram
25 scf(3); bode(sys_cl);

```

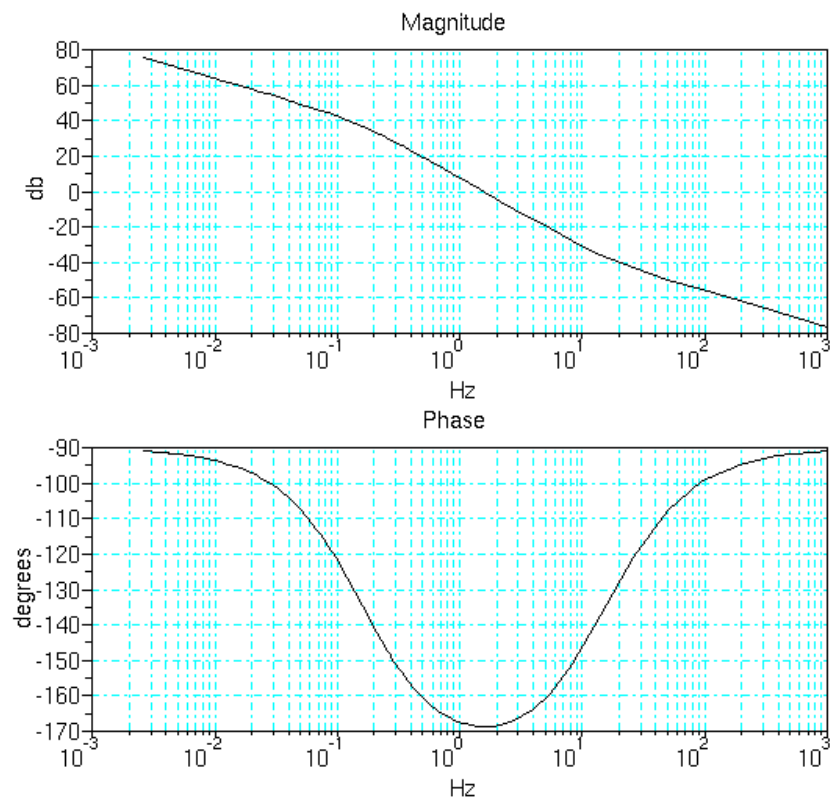


Root locus and Bode plots with ScicosLab

PLANT



PLANT + CONTROLLER

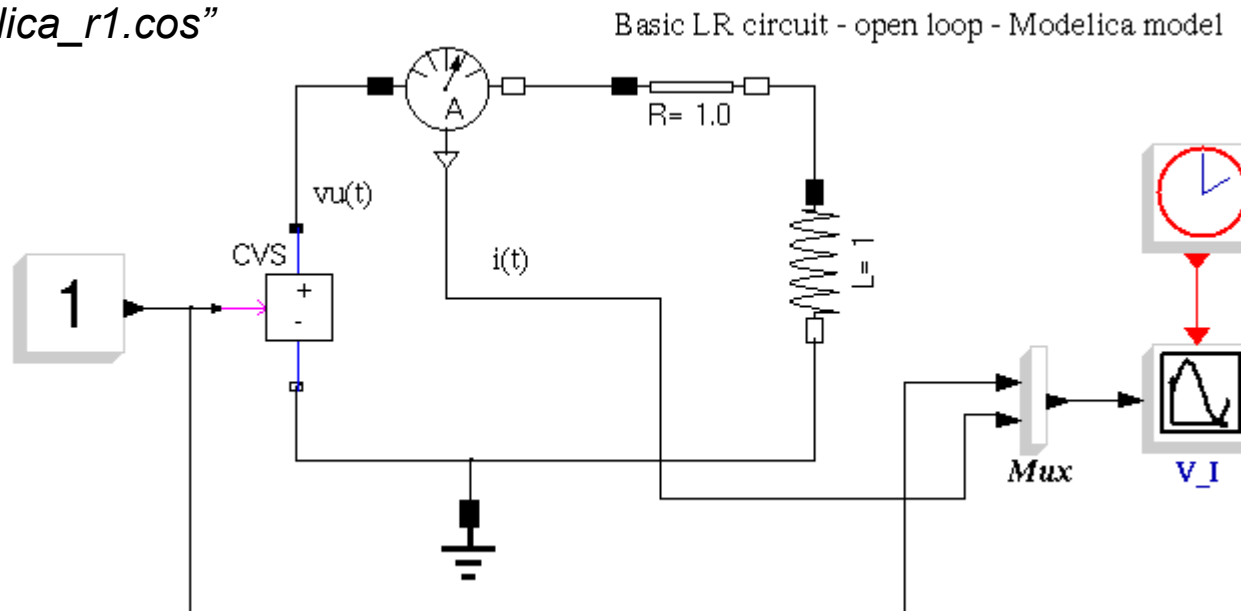


Mathematical models: an easy example.

Consider the basic LR circuits show below.

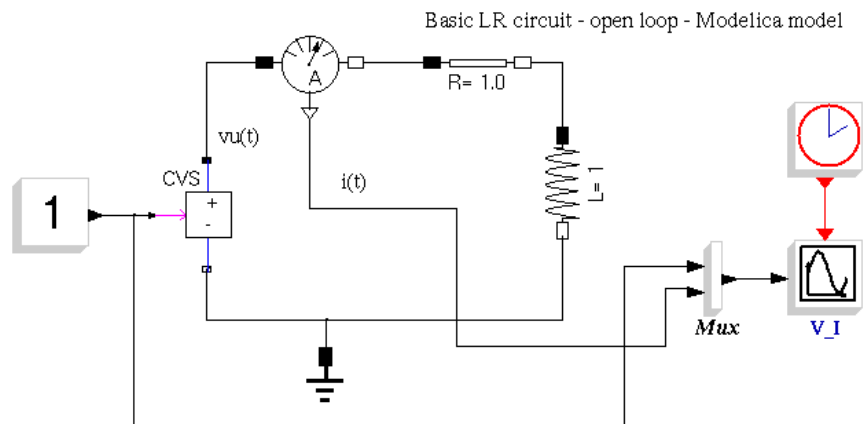
You would like to control the current “ $i(t)$ ” that flows inside the inductance using “ $v_u(t)$ ” (the voltage source that drive the coil).

“*lr_cc_modelica_r1.cos*”



Basic modelling

The equations are:



$$v_u(t) = R \cdot i(t) + \frac{d}{dt} \phi(t)$$

$$\phi = L \cdot i$$

$$v_u(t) = R \cdot i(t) + L \frac{d}{dt} i(t) + i(t) \frac{d}{dt} L(t)$$

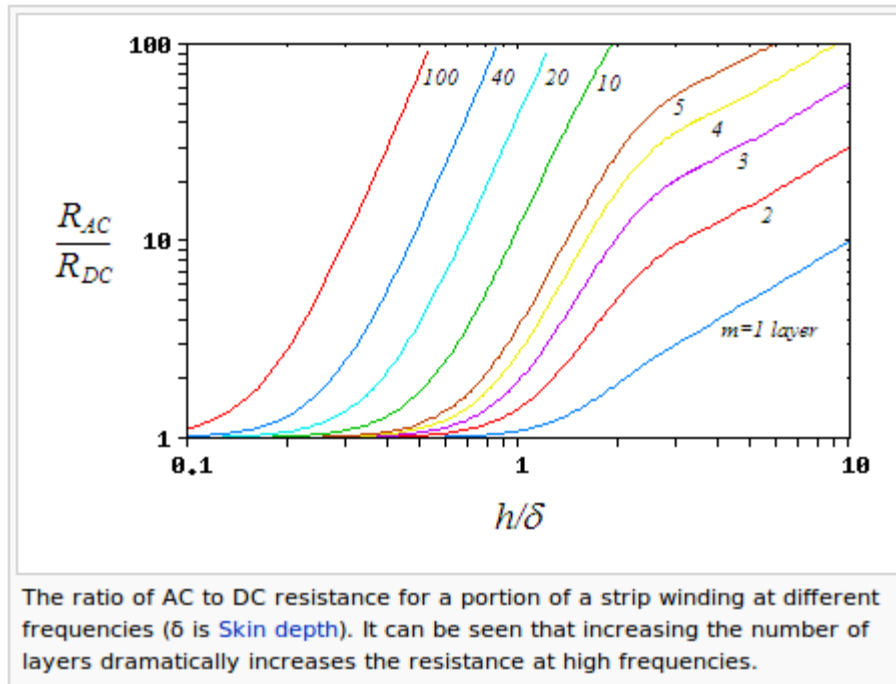
$$v_u(t) = R \cdot i(t) + L \frac{d}{dt} i(t)$$

$$L = \mu \cdot K \cdot \frac{N^2 A}{l} \quad \mu = \langle \text{material} \rangle$$

$$K = f_{Nagaoka} \left(\frac{d}{l} \right)$$

Basic modelling: simple but not simpler

Also R has some “time” dependencies



$$v_u(t) = R \cdot i(t) + L \frac{d}{dt} i(t)$$

$$L = \mu \cdot K \cdot \frac{N^2 A}{l}$$

Skin effect

$$\delta = \sqrt{\frac{2\rho}{\mu\omega}}$$

Proximity effect (multi layer solenoid) + skin effect

Why do you need discrete (sampled) systems?

The time sampling signal technology was born (1930, Bell) to improve the communication capability of telephone networks (more conversations on the same wired or radio connection).

No one suggested – at the time - sampled (discrete) feedback systems.

Sampled systems were considered an “aberration”.

Why sampling a signal when:

- the sensors produce *continuous signals*;
- the actuators need *continuous signals*;
- all the available *computing blocks* were based on analog (mechanical, hydraulic or electrical) *continuous time* technologies.

- 1939 -

Q. Why do you need discrete (sampled) systems?

A. Because there was a war.

If you need to point a cannon to an aircraft you can use a RADAR as sensor BUT all the RADAR signals (distance, bearing, etc.) are intrinsically “sampled” (discrete in time).

- 2010 -

Q. Why do you need discrete (sampled) systems?

A. Because there is the PC.

The personal computer is fully based on digital technology, in which all the signals are sampled in time (discrete in time, sampling) and in amplitude (discrete in value, quantization).

- 2010 -**Q. Why do you need discrete (sampled) systems?
A. Because there is a PC with ScicosLab**

All the software simulators - running on a PC - model continuous systems using equivalent discrete systems.

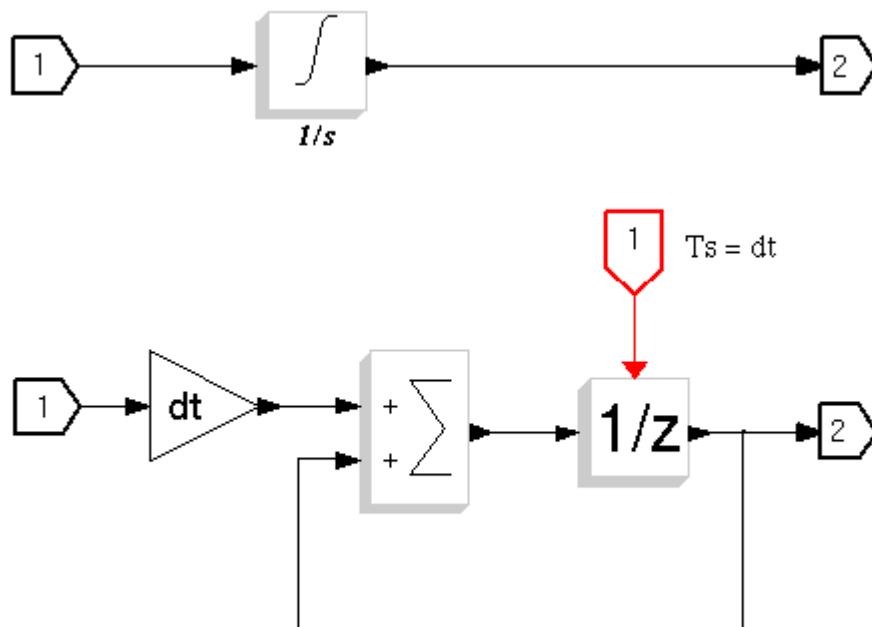
Three questions:

How is it possible to build a discrete equivalent of a continuous system?

It is (the discrete equivalent) just an approximation of the continuous one or a completely different “thing”?

What are the hypothesis and the limits of this “emulation”?

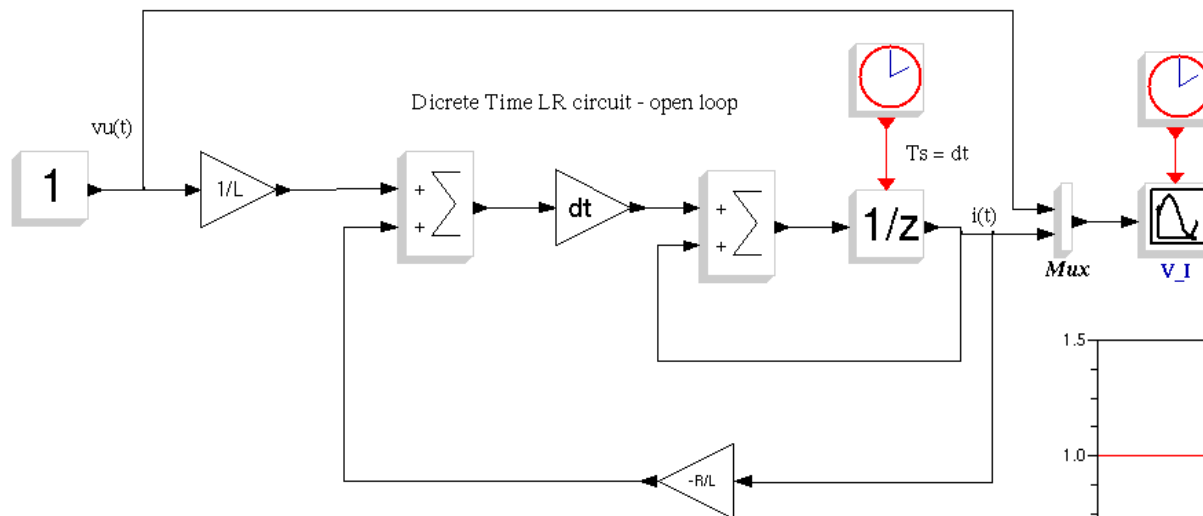
The discrete integrator (according to Euler)



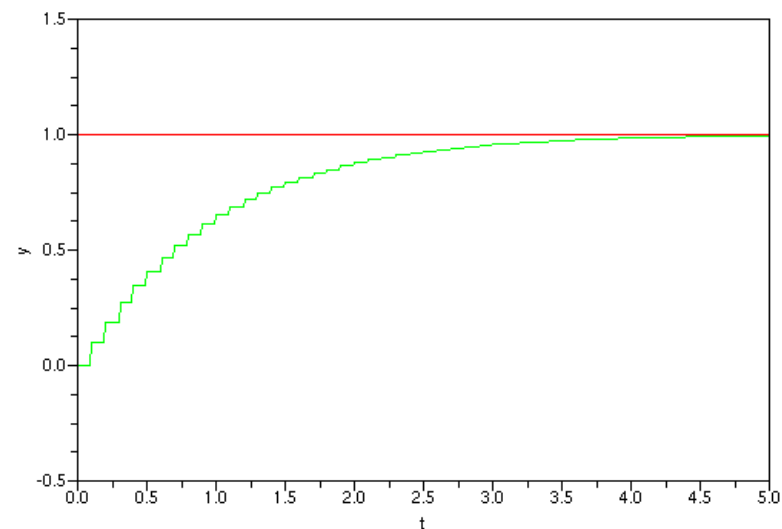
1957 stamp of the former Soviet Union commemorating the 250th birthday of Euler. The text says: 250 years from the birth of the great mathematician, academician Leonhard Euler.

Open Loop discrete Scicos simulation

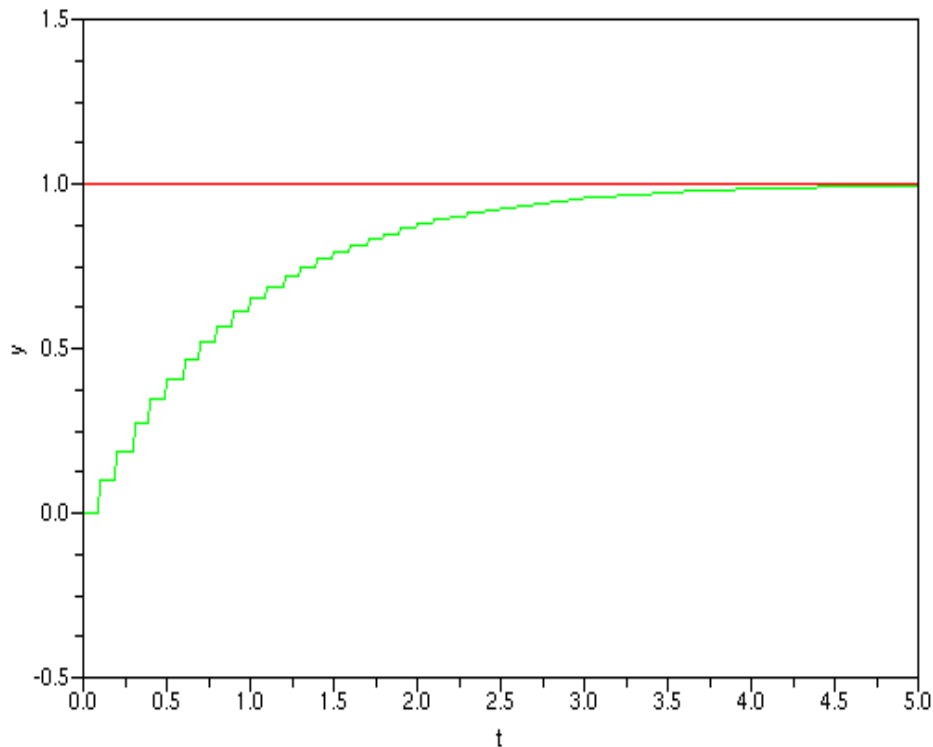
Using the previous transformation the result is:



"lr_cc_ol_r1.cos"



Open Loop discrete current controller simulation



Observations and open discussion:

The same results.

How have you calculated T_s ?

What happens if you change T_s ?

Can you show the discrete nature of the system?

Does Scicos works using this technique ?

Hybrid systems

Why do you have fallen into hybrid systems?

Because - most of the time - the PLANT is a continuous time system and the CONTROLLER is realized with digital technology (discrete time system).

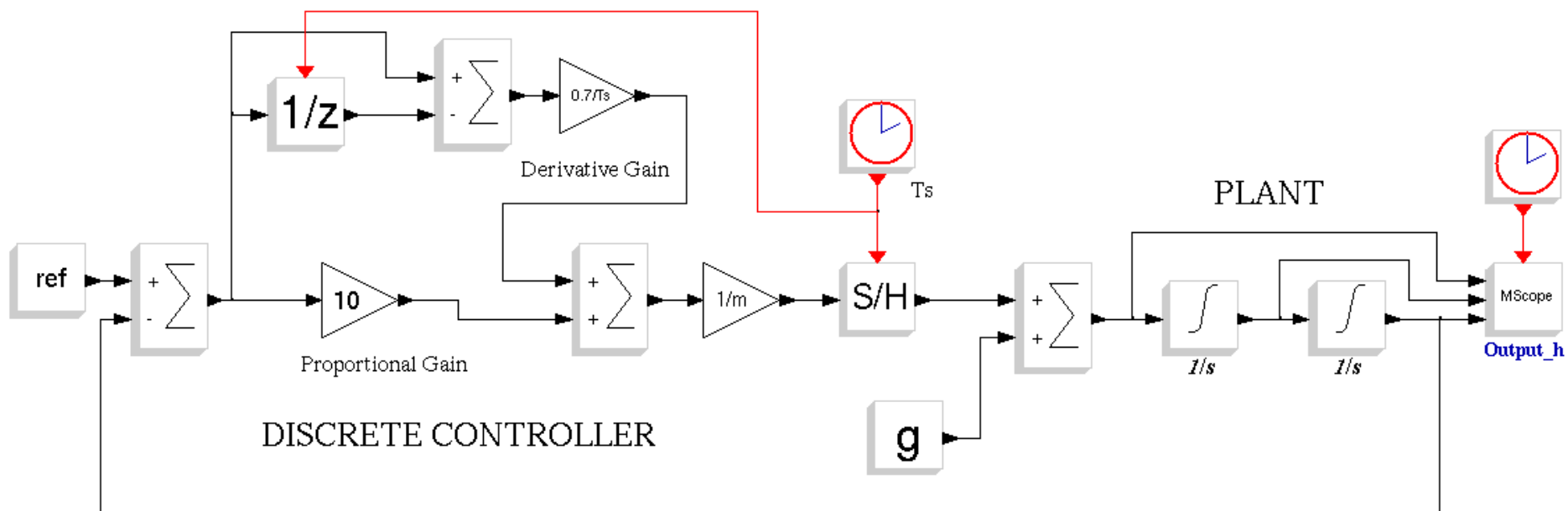
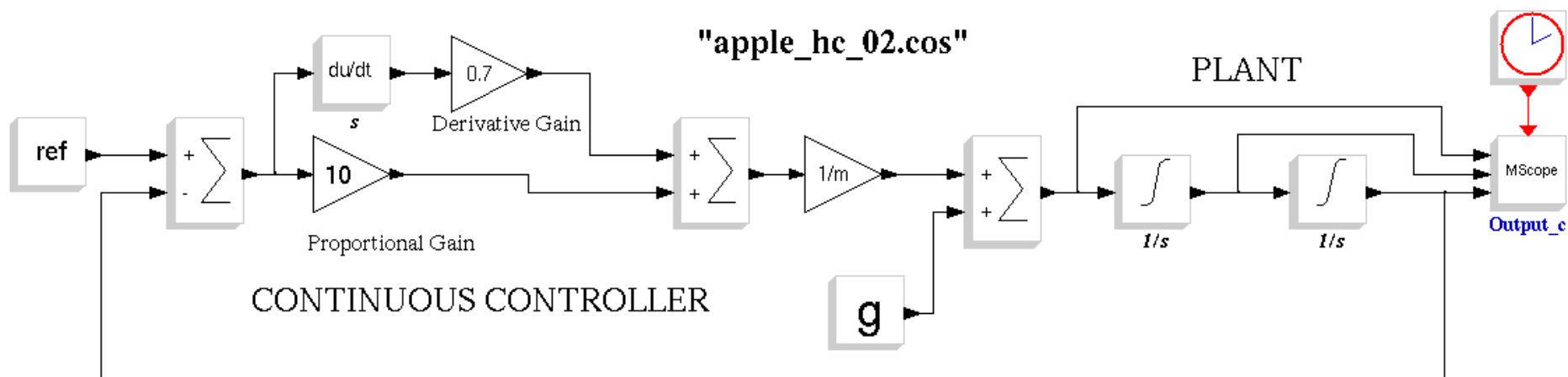
Why digital technology? Because 1000 low quality transistors are cheaper than 1 high quality device. Digital controllers are becoming less expensive than their analog equivalents. Moreover, digital control offers capability well beyond the usual analog PID(networking, supervision, diagnostic, etc.).

Digital control is a perfect solution for an Internet connected world.

Where is the weak point?

Software.

Continuous vs Discrete controller



Continuous vs Discrete controller

Observations and open discussion:

It works but the response is a bit different ...

Where is the input sampler ?

Where are the input and the output filters ?

Why is the D gain different and the P gain the same ?

What happens if you change T_s ?

Is digital control cost effective also for very simple systems?

Is digital control “safe” ?

How to design discrete controllers for continuous plant

Old way (design in “s” space):

- You build a model for the PLANT
- You design a controller using continuous time tools (Laplace, root locus, Bode)
- You choose “Ts”
- You transform the continuous controller in a discrete one (Tustin, etc.)

New way (design in “z” space):

- You build a model for the PLANT
- You choose “Ts”
- You transform the continuous plant in a discrete equivalent
- You design a controller using discrete time methods (deadbeat, pole placement, etc.)

Plus and minus of the two design methods

Old way advantages:

- You can continue to use familiar tools and methods (e.g. Bode plots)
- You can postpone the decision about “Ts” (critical decision)

Disadvantages:

- No better performances compared to the “analog” design.

New way advantages:

- You can use high quality transformations to create a discrete equivalent of the PLANT. This improve the design of the controller without additional costs.
- You can use control topology too expensive for analog (c.t.) implementation

Disadvantages:

- You need to choose Ts early (critical decision)
- You need to master discrete design techniques

Do not push too hard on Ts !

Beware! You must avoid the temptation to push too much the sampling frequency.

If you double the sampling frequency, you double the required computational requirement, so the power consumption, so the weight, so the cost....

If you push too much the sampling frequency, you aggravate the problems that derive from the finite mathematical precision. You risk to spend time to accumulate “zero” quantity or differentiate equal values with very small denominators.

Finally, you become too much sensitive to numerical (finite precision) and signal (real signal) noises, in other words, avoid “zipping” poles in (1,0)

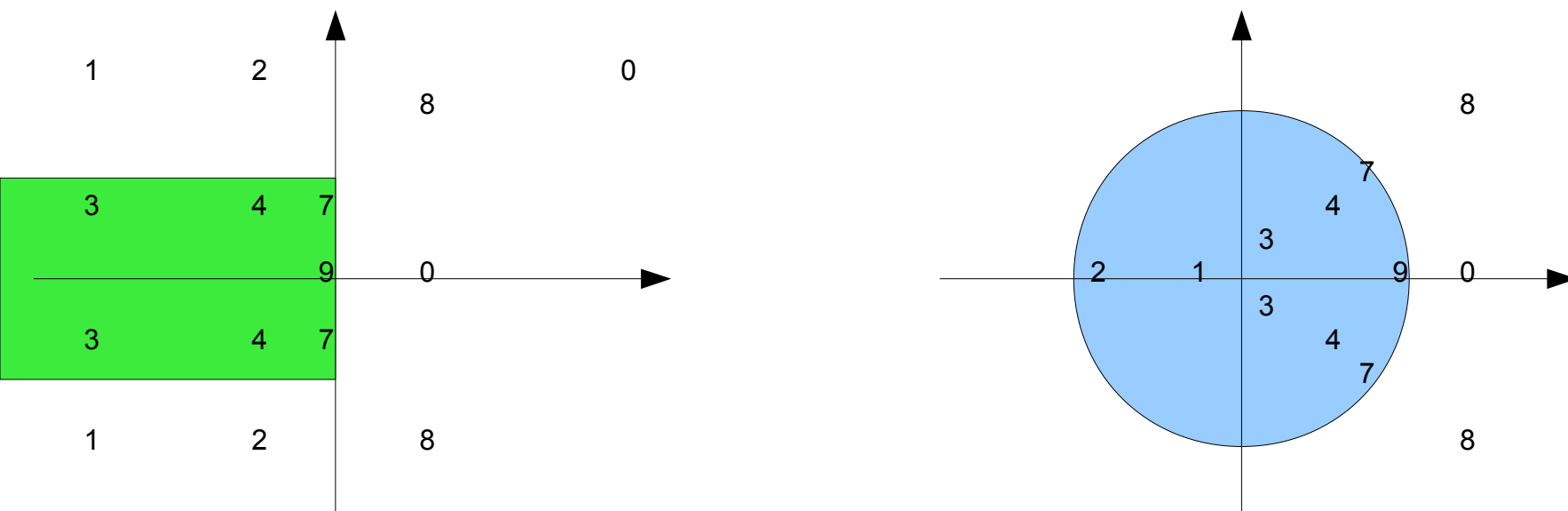
$$z = e^{sT_s}$$

Small Ts means $z \rightarrow (1,0)$ not matter the value of “s”.

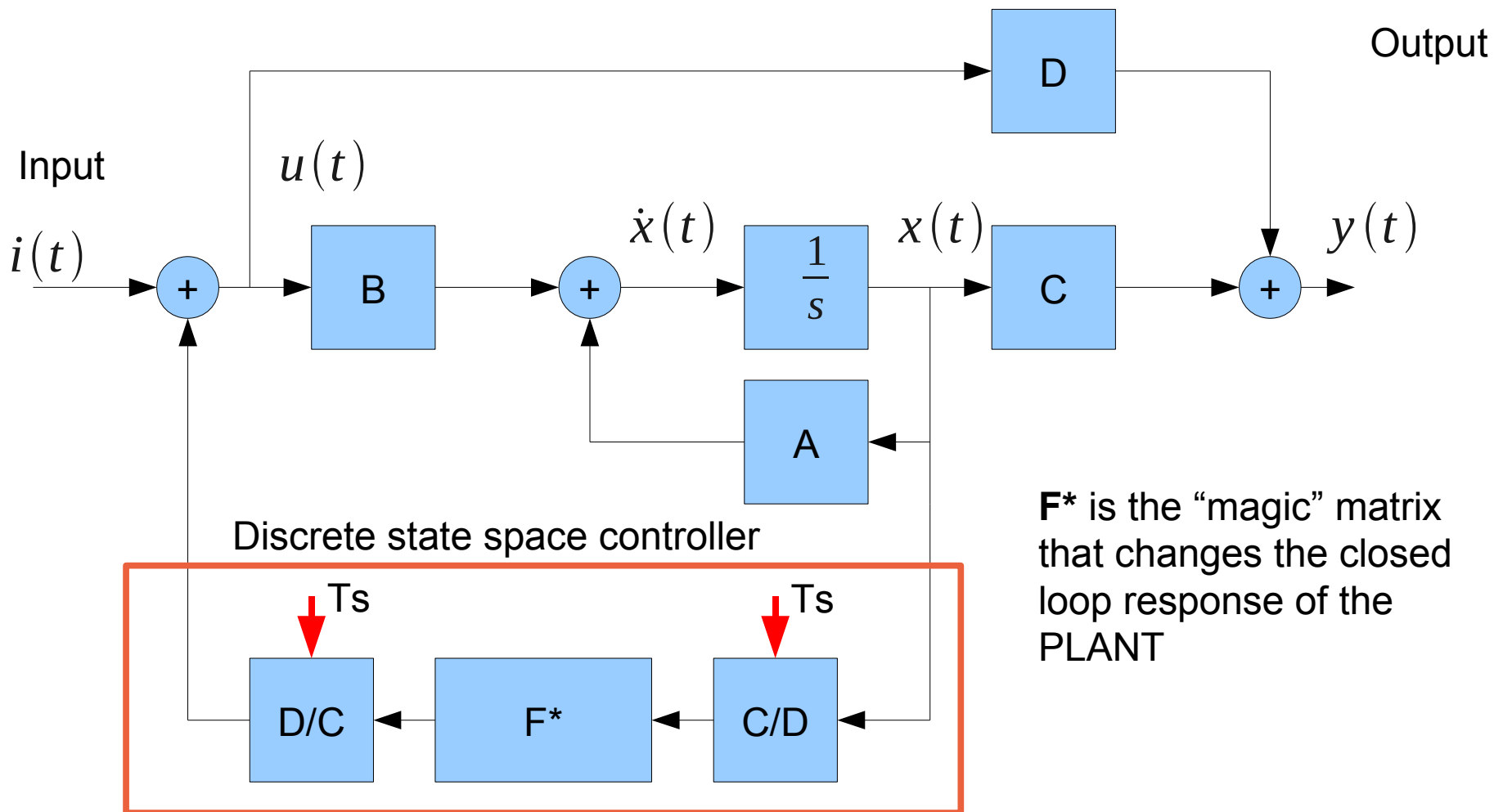
L and Z transforms intimate relationship

Think back to our original problem to create a discrete system that “emulates” a continuous one. Let us suppose that we have already designed our complete system, so we know where the poles are in the “s” (Laplace) plane.

$$s = \sigma + j\omega \quad ; \quad z = e^{sT_s}$$



Hybrid state space control



Scicos / Kepler integration

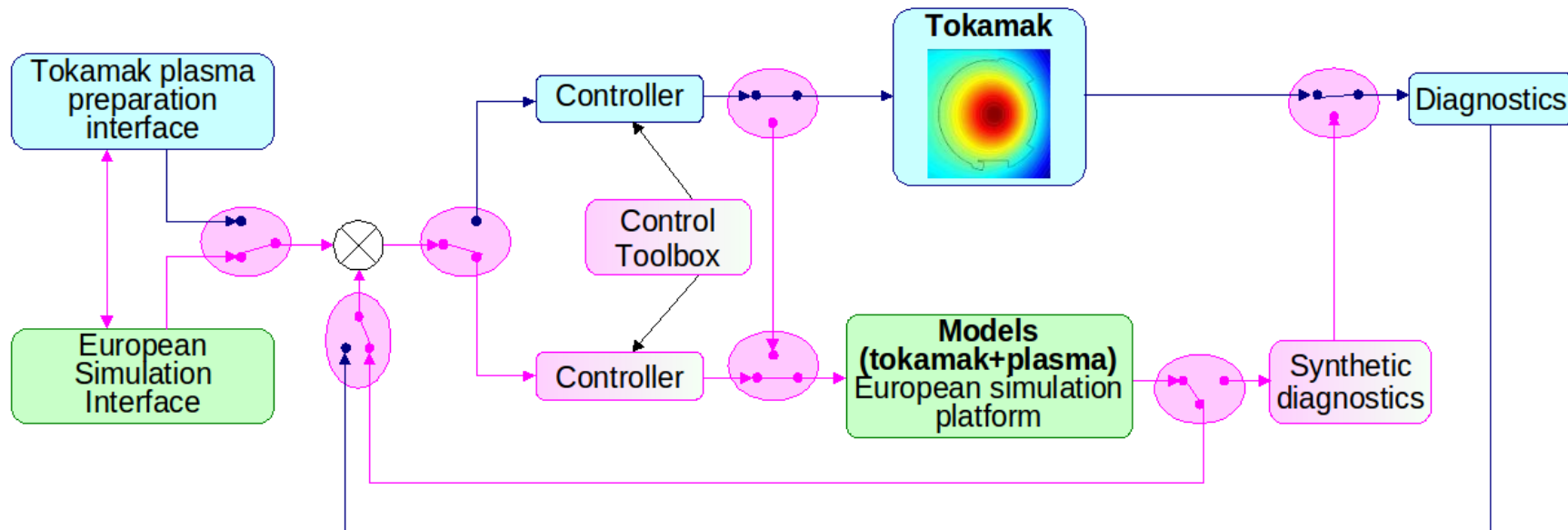
Introduction


Introducing ScicosLab and Kepler


Plasma discharge flight simulator




Association
Euratom-Cea



 Common architecture
for plasma operation

 ITM tools under
development

 To be developed for
the flight simulator

➔ Switches allow to select the flight simulator mode of operation (functionalities)

ScicosLab and Kepler together

Kepler

Flexible integration of complex physical simulations developed using other platforms (FORTRAN, C, Matlab, R, etc.)

Strategic choice of ITM

ScicosLab / Scicos

Dedicated platform for design and simulation of complex control systems

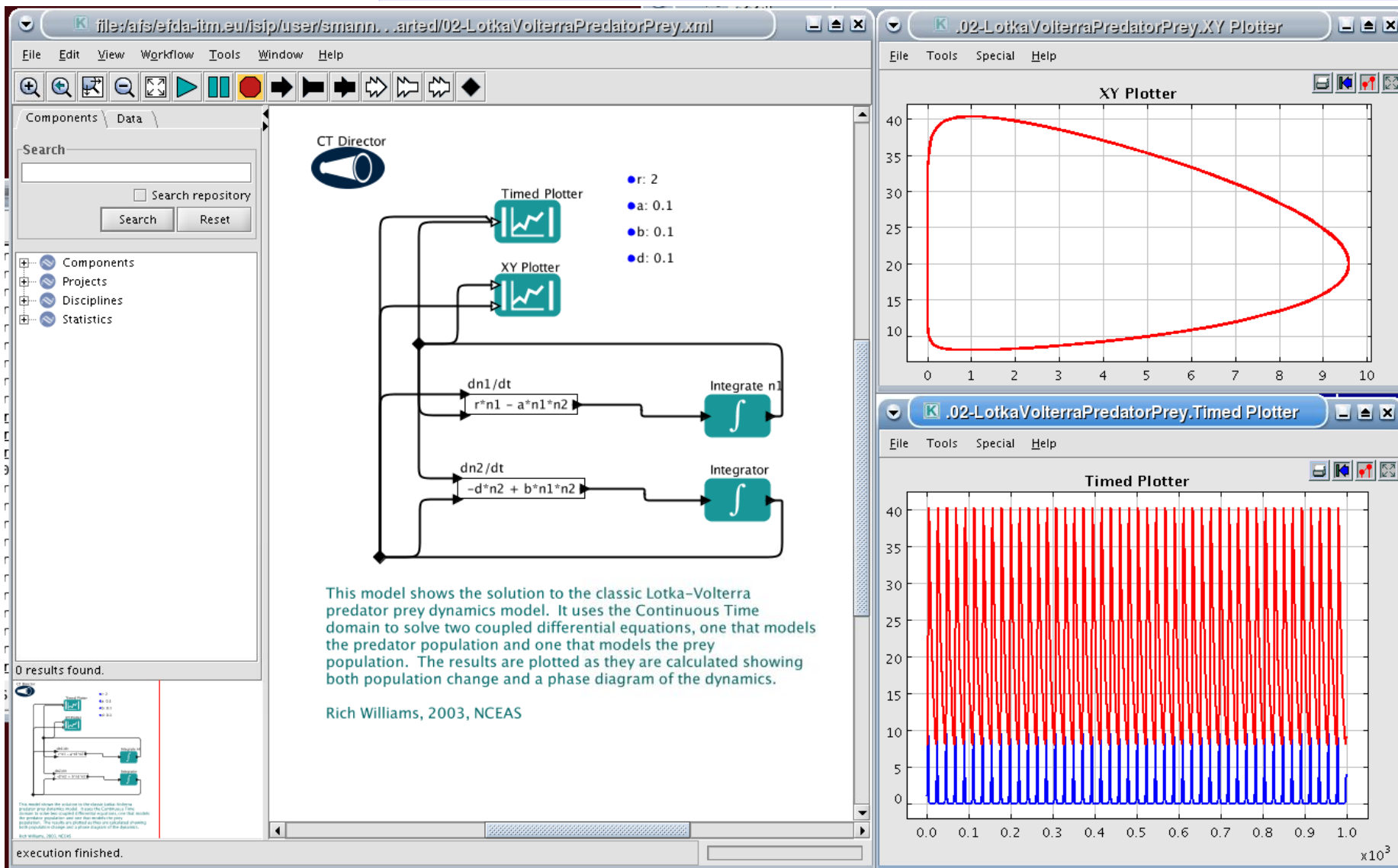
Built in code generation capabilities for simulation and application on embedded systems

Simulink, ScicosLab, Kepler

	Simulink	Scicos	Kepler
Main entity	Diagram	Diagram	Work flow
Atomic entity	Block (C)	Block (C, Scilab)	Actor (Java)
Sub assembly	SubDiagram	SuperBlock	Composite Actor
Connection	Link (line)	Link	Relation
Script language	Matlab (*.m)	Scilab (*.sci)	Not Available
Code Generation	Real Time Workshop	Scicos Code Generators	Not available
	<i>The Bad</i>	<i>The Ugly</i>	<i>The good</i>

Kepler

Introducing ScicosLab and Kepler



ScicosLab

Introducing ScicosLab and Kepler

The screenshot displays the ScicosLab environment with several windows:

- ScicosLab (Main Window):** Shows the ScicosLab-4.4b7 version information and startup execution details. CPU time is 43.612 seconds.
- Scipad 8.39BP1 - Lorenztz.sce:** Contains the following code:

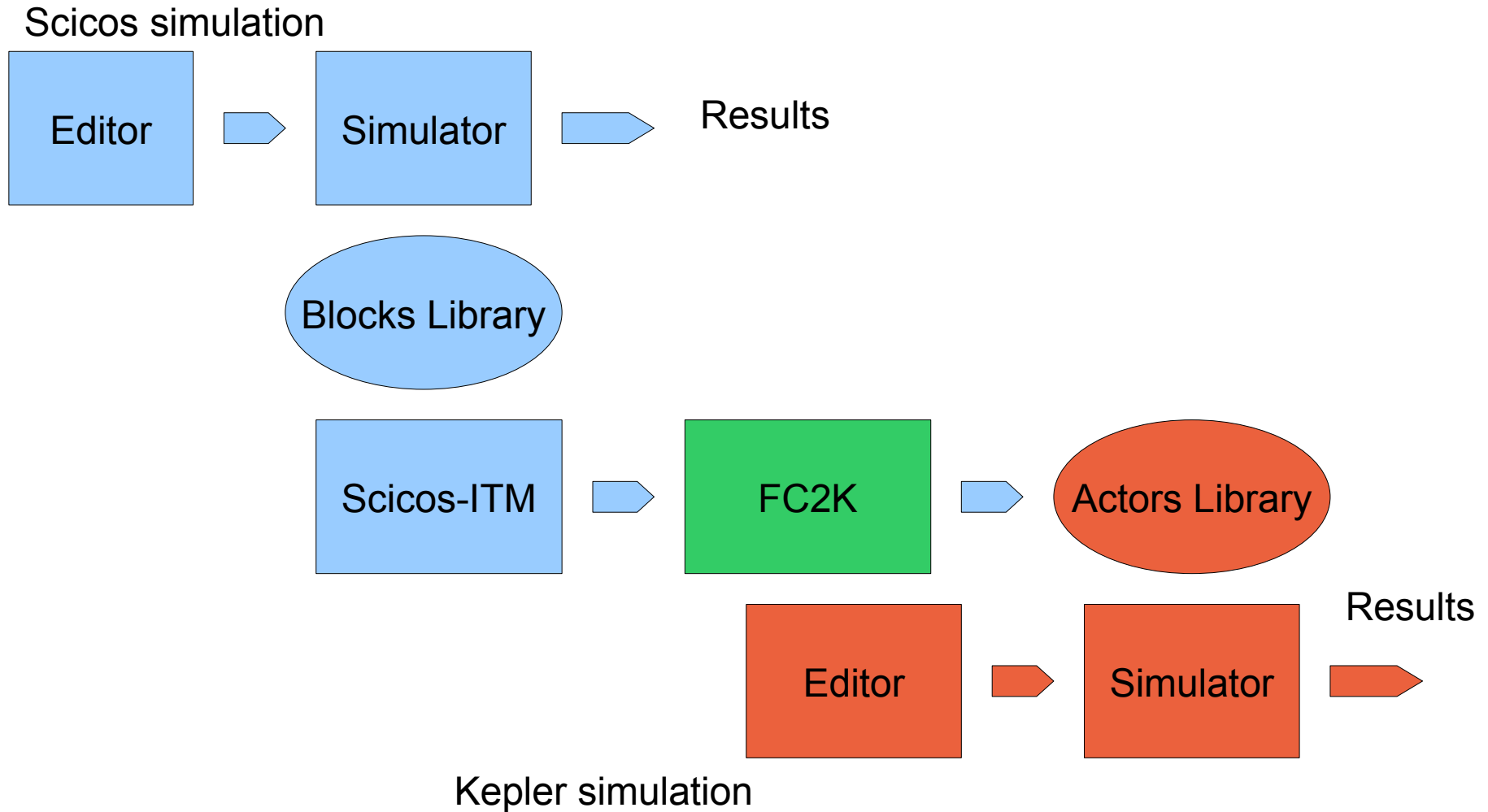

```

1 //set sampling time
2 Tsampl=3e-3
3
4 //set parameters
5 a=10
6 b=28
7 c=8/3
8
9 //set initial conditions
10 ci1=[5.5;5.49;5.51;5.511]
11 ci2=[5;4.99;5.01;5.011]
12 ci3=[20;19.99;20.01;20.011]
13
14 //set colors for scopes
15 vcol=[0;3;5;9]
16
17 //set end simulation time
18 Tfin=30
      
```
- Lorenztz [edited]:** A block diagram of the Lorenz attractor system, featuring three integrators (1/s), gain blocks (a, b, c), and summing junctions (+, -).
- ScicosLab Graphic (20018):** A 2D plot of the Lorenz attractor.
- ScicosLab Graphic (20016):** A 3D plot of the Lorenz attractor.
- 2D Scope:** Three vertically stacked plots (Graphic 1, 2, 3) showing the time evolution of the system's variables.
- System Monitor:** A vertical panel on the right showing system performance metrics like CPU usage, memory, and network activity.

Scicos-ITM How-To

- 1. Design the control system in ScicosLab**
- 2. Prepare the controller for Code Generation**
- 3. Generate the code using Scicos-ITM**
- 4. Generate a Kepler actor using FC2K**
- 5. Insert the actor inside the Kepler work flow**
- 6. Run the Kepler simulation**
- 7. Don't worry. Be happy :-)**

From ScicosLab to Kepler



The system: the floating apple

- I have an apple at $y(0)=y_0=1.0\text{m}$.
- At $t=0$ I drop the apple, and the apple falls down.
- I'm not satisfied: I'd like to see the apple floating at a reference height (ref=0.5m).
- I need a controller to implement a closed loop feedback system.

Open Loop “PLANT” model

The “PLANT” : a free falling apple

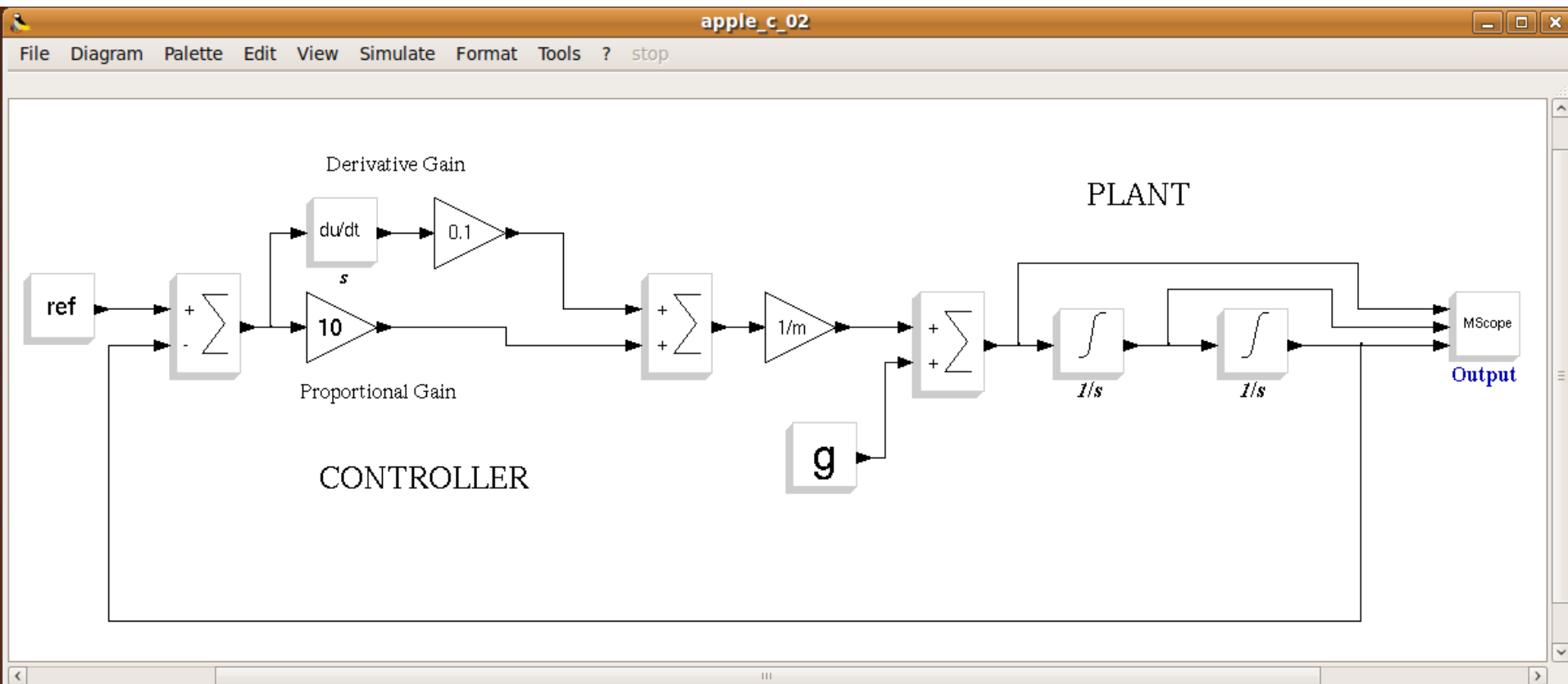
$$F = G \frac{m_a m_T}{r^2}$$

$$F = m a \quad a = \frac{F}{m} \quad a_a = G \frac{m_T}{r^2} ; g = -9.81 \text{ m/s}^2$$

$$v(t) = v_0 + \int a(t) dt \quad y(t) = y_0 + \int v(t) dt$$

$$y(t) = y_0 + \frac{1}{2} g t^2$$

Time continuous controller



Equivalent discrete system (plant + controller)

Using Euler, we create a discrete equivalent system

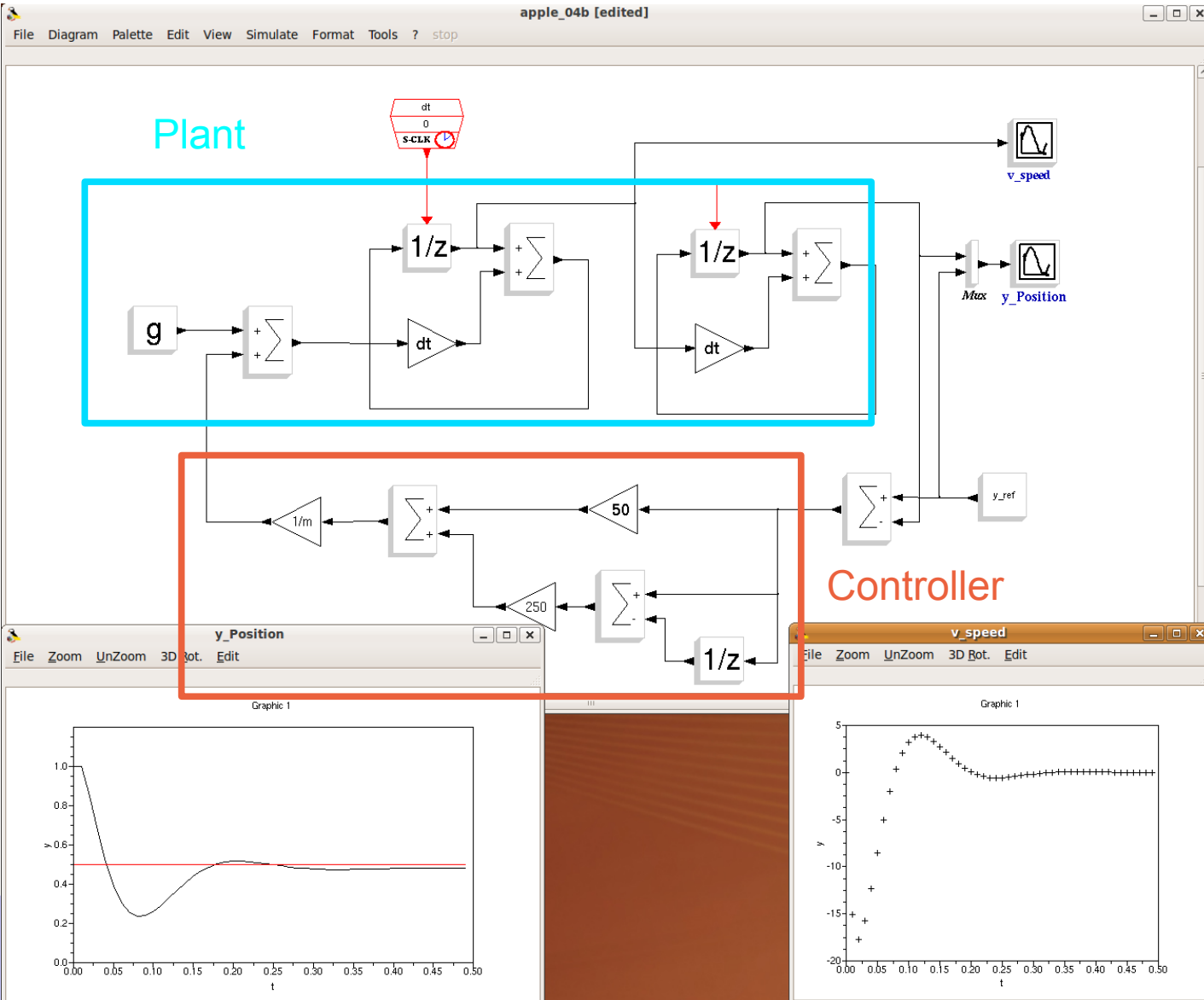
$$v(t) = v_0 + \int a(t) dt \quad ; \text{ integral (differential) equation}$$

$$v_{k+1} = v_k + a_k \text{ delta} \quad ; \text{ difference equation}$$

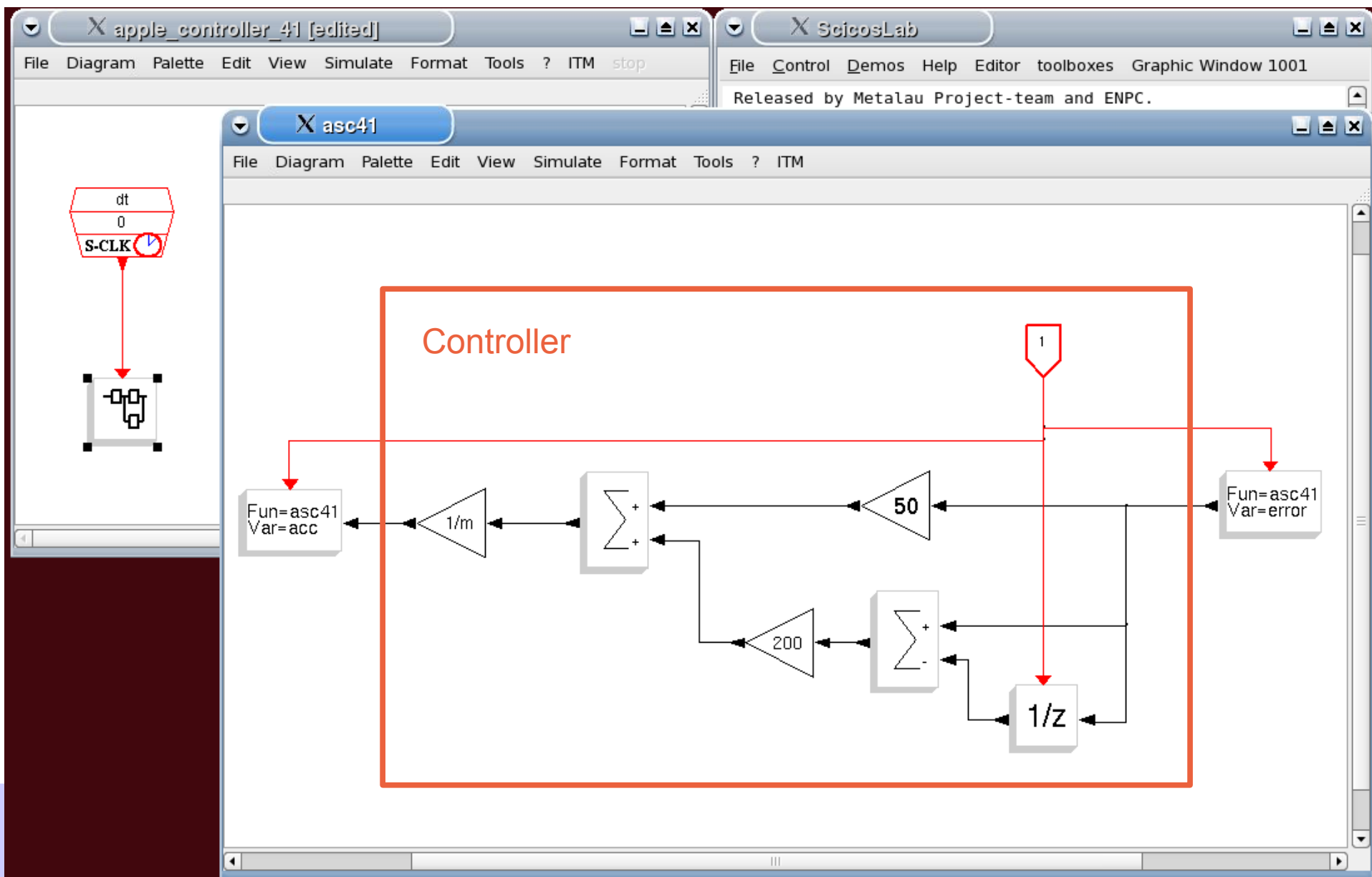
$$v_k = v(kT_s) \quad ; \text{ time sampling}$$

$$\text{delta} = T_s \quad ; \text{ discrete time = sampling time}$$

1. Design the control system in ScicosLab



2. Prepare the controller in ScicosLab for code generation



Introducing ScicosLab and Kepler

3. Generate the code using Scicos-ITM

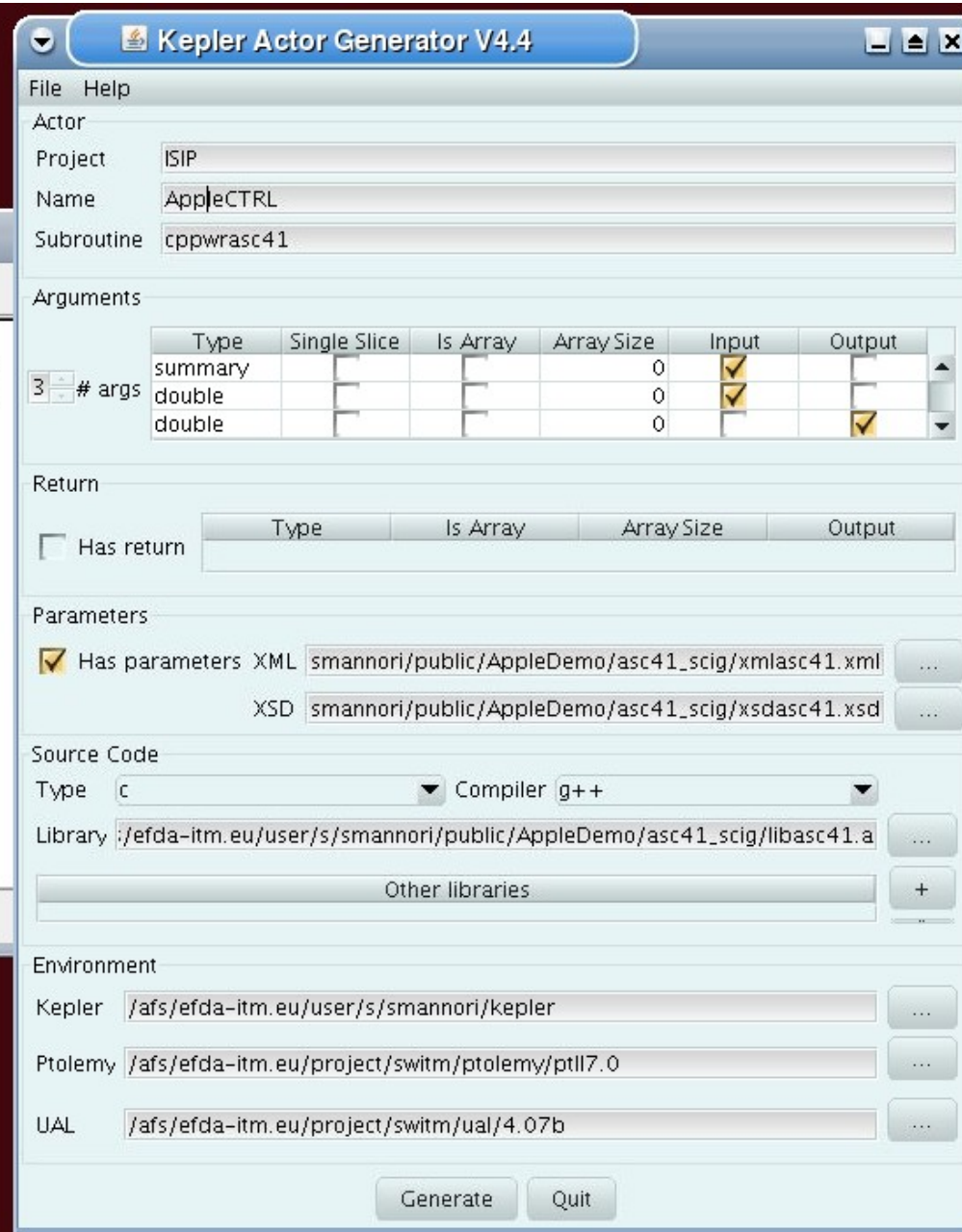
The screenshot displays the ScicosLab environment. The main window, titled 'asc41 [edited]', shows a control block diagram. A red box labeled '1' is connected to several blocks: a gain block '1/m', a summing junction, a gain block '50', a summing junction, a gain block '200', a summing junction, and a block '1/z'. There are also two function blocks labeled 'Fun=asc41 Var=acc' and 'Fun=asc41 Var=error'. A terminal window at the bottom shows the compilation process:

```
total 9436
-rw-r--r-- 1 smannori isip 15990 Feb 23 11:15 asc41.c
-rw-r--r-- 1 smannori isip 1074 Feb 23 11:15 asc41_Cblocks.c
-rw-r--r-- 1 smannori isip 5704 Feb 23 11:15 asc41_Cblocks.o
-rw-r--r-- 1 smannori isip 26472 Feb 23 11:15 asc41.o
-rw-r--r-- 1 smannori isip 629 Feb 23 11:15 common.c
-rw-r--r-- 1 smannori isip 6136 Feb 23 11:15 common.o
-rw-r--r-- 1 smannori isip 527 Feb 23 11:15 cppwrasc41.cpp
-rw-r--r-- 1 smannori isip 1769720 Feb 23 11:15 cppwrasc41.o
-rw-r--r-- 1 smannori isip 970 Feb 23 11:15 cwrasc41.c
-rw-r--r-- 1 smannori isip 4256 Feb 23 11:15 cwrasc41.o
-rw-r--r-- 1 smannori isip 7842708 Feb 23 11:15 libasc41.a
-rw-r--r-- 1 smannori isip 1177 Feb 23 11:15 Makefile
-rw-r--r-- 1 smannori isip 10 Feb 23 11:15 xmlasc41.xml
-rw-r--r-- 1 smannori isip 10 Feb 23 11:15 xsdasc41.xsd
<smannori@enea142 ~/public/AppleDemo/asc41_scig>
```

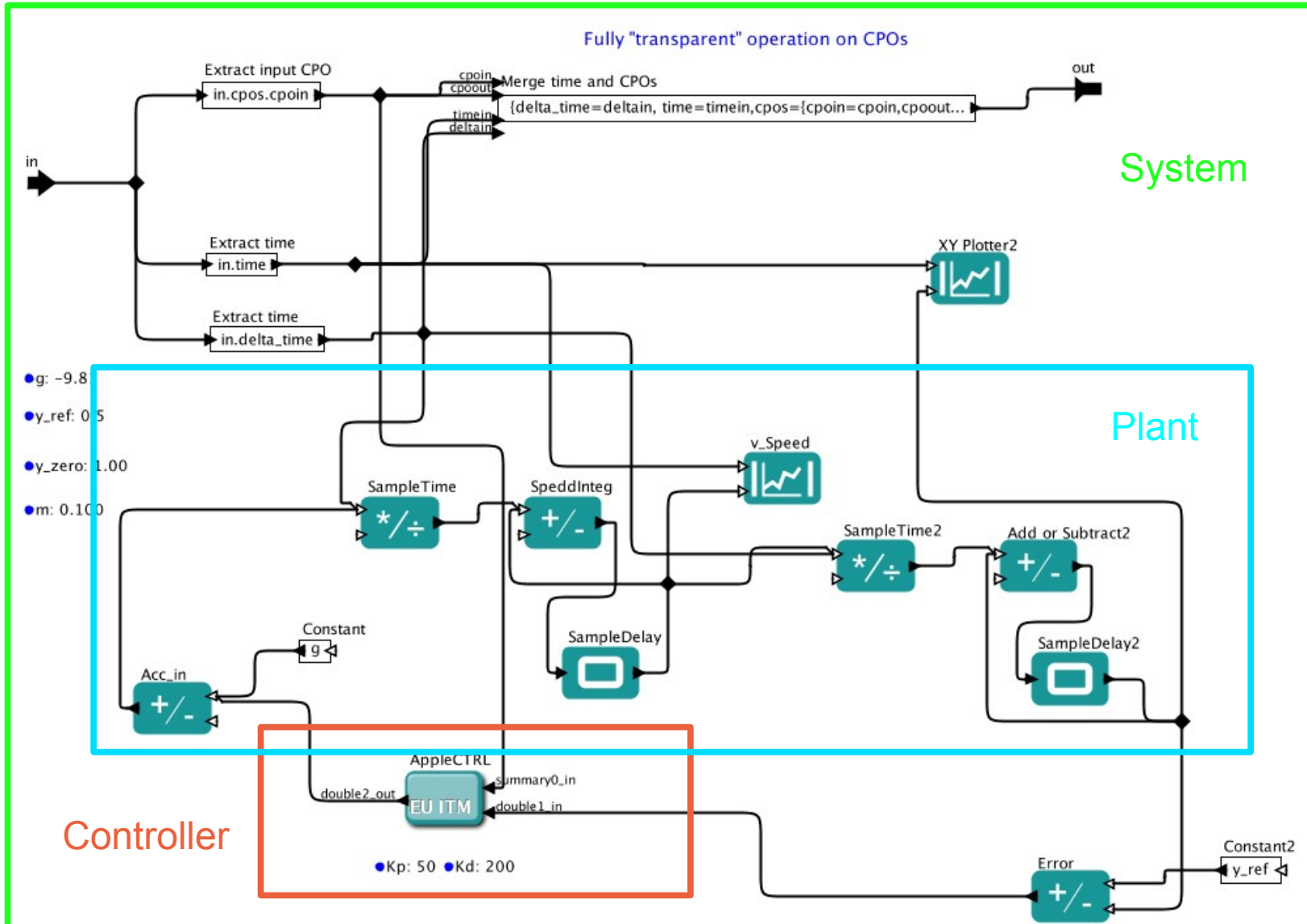
4. Generate a Kepler actor using FC2K

```
asc41.c
asc41_Cblocks.c
asc41_Cblocks.o
asc41.o
common.c
common.o
cppwrasc41.cpp
cppwrasc41.o
cwrasc41.c
cwrasc41.o
libasc41.a
Makefile
xmlasc41.xml
xsdasc41.xsd
>fc2k
```

```
actors
```

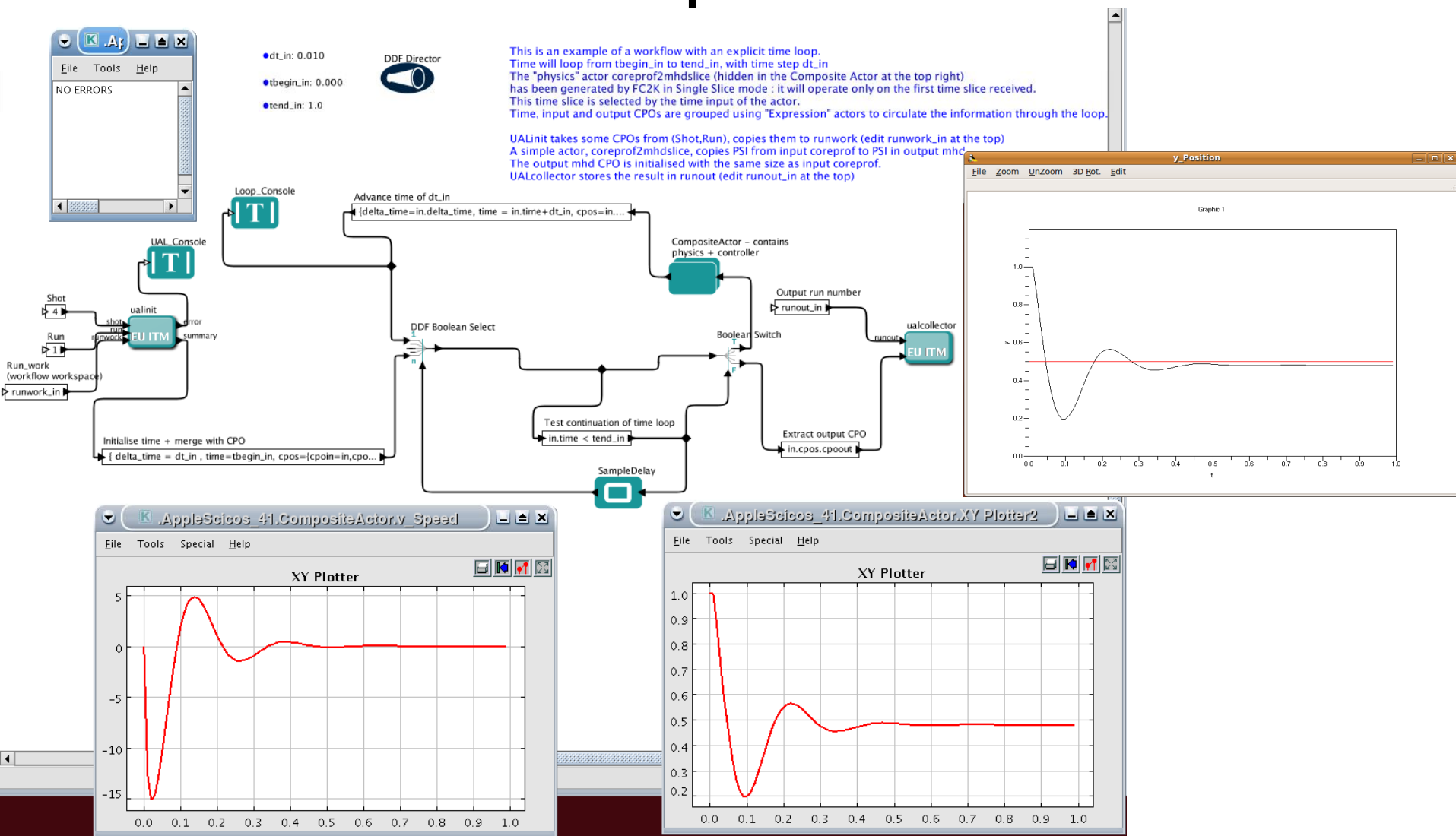


5. Insert the actor inside the work flow



Introducing ScicosLab and Kepler

6. Run the Kepler simulation



7. Don't worry. Be happy :-)

