USING SCILAB/SCICOS WITH RTAI-LAB

Roberto Bucher
University of Applied Sciences of Southern Switzerland
Galleria 2, Manno 6928, C H
roberto.bucher@supsi.ch

1 Introduction

Computer Aided Control System Design (CACSD) subsumes a broad variety of computational tools and environments for control system design, real time simulation, with and without hardware in the loop, making the best use of high desktop computer power, graphical capabilities and ease of interaction with low hardware cost. Integrated CACSD software environments allow an interactive control system design process to be automated with respect to multi-objective performances evaluation and multi-parameter synthesis tuning. Visual decision support provides the engineer with the clues for interactively directing an automated search process to achieve a well balanced design under many conflicting objectives and constraints. Local/remote on line data down/upload makes it possible a seamless interaction with the control system, in order to supervise its operation and to adapt it to changing operational needs.

2 Installing the RTAI add-ons for Scilab/Scicos

2.1 Installing RTAI-Lab

2.1.1 Getting the files and the libraries

First of all you need RTAI-Lab. RTAI-Lab is an open project integrated in the Linux RTAI distribution. In order to run RTAI-Lab you need the following packages:

Linux RTAI download the last stable release from the RTAI homepage www.rtai.org

Mesa libraries download this libraries using the following steps:

1. Download MesaLib-6.2.tar.gz (from www.mesa3d.org) in a temporary directory (/tmp)
2. Untar the archive : tar xvzf MesaLib-5.0.1.tar.gz
3. cd /tmp/Mesa-6.2
4. make linux-86
5. make install
   - /usr/include + /usr/lib
6. make install (
   - /usr/X11R6/include + /usr/X11R6/lib

Compile and install the EFLTK package

1. Download EFLTK from CVS in a temporary directory (/tmp)
   cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ede login
   (press ENTER when CVS asks for password)
   cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/ede co efltk
2. cd /tmp/efltk
3. ./build.gcc –prefix=/usr/local –enable-xdbe –enable-opengl –enable-threads
4. make install
5. /sbin/ldconfig

2.1.2 Installing RTAI and RTAI-Lab

Follow these steps in order to install Linux RTAI

- Kernel
  1. Get a "vanilla" kernel from www.kernel.org
  2. Unpack the kernel
  3. Unpack Linux RTAI
  4. Patch the kernel with the adeos patch
     (patch -p1 < <rtaidir>/base/arch/i386/patches/<kernel-version>.patch)
  5. make xconfig
  6. configure the kernel
  7. make bzImage
  8. make modules
  9. make modules_install
  10. make install
  11. fit lilo or grub for this new kernel
- install COMEDI : install comedi and comedilib from www.comedi.org
- install RTAI
  1. Go to the RTAI directory
  2. make xconfig
  3. Check "COMEDI support under LXRT" in the Add-ons submenu and give the directory where you installed comedilib
  4. Check "RTAI-Lab" under the RTAI-Lab submenu and add the directory where you installed the efltk libraries (normally "/usr/local")
  5. save and exit from xconfig
  6. run "make" and "make install"
- IMPORTANT: Add "/usr/realtime/bin" to the PATH variable in "/etc/profile" or ".bashrc".

2.1.3 Installing Scilab

The first step is to download and install the full source Scilab-3.0 release from "www.scilab.org"; it contains Scicos already. Install scicos:

1. Go in the directory "/usr/local"
2. Unpack Scilab
3. cd scilab-3.0
4. ./configure
5. make all

2.1.4 Installing the RTAI add-ons for Scilab

Now follow these steps to properly install all the Scilab/Scicos add-ons for RTAI-Lab:

1. become superuser
2. go in the "macros" directory found here
3. modify eventually in the file "Makefile" the line

   ```
   SCILAB_DIR = /usr/local/scilab-3.0
   ```

   to fit your SCILAB installation
4. run "make install"

Each user who want to work with the Scilab/Scicos RTAI add-ons has to modify his own ".scilab" file. This operation can be done running as normal user "make user" in this "macros" directory. This command add the following lines to the user ".scilab" file:

```
load('SCI/macros/RTAI/lib')
%scicos_menu($+1)=['RTAI','RTAI CodeGen']
scicos_pal($+1,:)=['RTAI-Lib','SCI/macros/RTAI/RTAI-Lib.cosf']
```

These lines add the menu "RTAI→CodeGen" to the scicos window and the new RTAI-Lib.cosf library with the RTAI specific blocks to the scicos palette.
At this point you are ready to generate code for RTAI, using the new menu "RTAI→RTAICodeGen" in the scicos window.

3 A simple example

3.1 Scheme

In the following, a simple example will be analyzed. The system is represented by a transfer function

$$Gs(s) = \frac{20}{s^2 + 4s}$$

with unity feedback,
The system has been implemented as discrete-time transfer function

$$Gz(z) = 10^{-6} \frac{9.987z + 9.973}{z^2 - 1.996z + 0.996}$$

with a sampling time of $1ms$. Different signals are sent to scopes, meters, and LEDs.
The model is saved with the name "test".

3.2 Implementation under Scilab

3.2.1 Designing the scheme

First of all we have to design the system using scicos. We can get the different blocks from the scicos palettes in order to obtain the desired system. By the next step we have to integrate some I/O into our scheme. We implemented three methods:

- I/O can be chosen from a specific RTAI Library (normal way)
- I/O were configured by hand (exceptional way)

These methods can be mixed together.

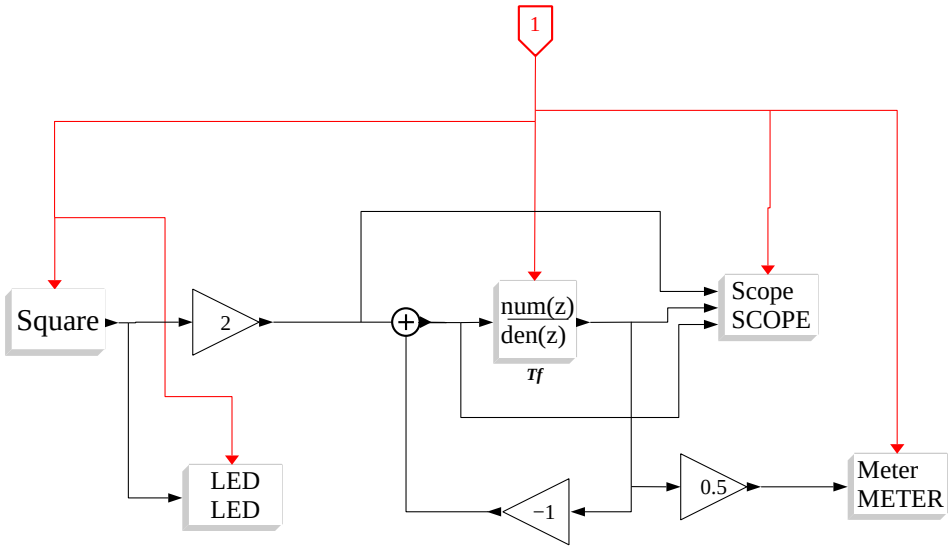

Figure 1: Scicos scheme

3.2.2 Implementing I/O using the RTAI palette

Figure 1 represents the Scicos scheme of the given example.
In order to generate the code this scheme must be transformed into a "Super Block" which can be used to generate the code. The menu "Diagram" "Region to Super Block" allows to select a part of the scheme and to put it into a "Super Block". We can access to the Super Block scheme simply by clicking on it. A good idea is to open the "Super Block" and to rename it ("Diagram" "Rename") to "test". This will be the name of the generated model and of the directory where the generated files are stored.
Now we can simply choose the menu "RTAI" "RTAICodeGen" to generate and compile the realtime code.
A dialog Box asks about some compilation parameters, in particular it propose the sampling time read from the "clock" event block.
After "OK" scicos performs the code generation and the compilation of the generated modules.

4 I/O Blocks

4.1 Basics

I/O blocks are implemented in a library (libsciblk.a). There are 3 kind of I/O blocks:

1. Blocks which can be connected to an input or an output port (ex. rtai_comedi_data)
2. Blocks which can be connected only to an input port (ex. step)
3. Blocks which can be connected only to an output port (ex. rtai_scope)

4.2 Available blocks

Figure 2 shows the palette with the present available RTAI blocks under Scilab/Scicos.

5 Implementation of new user blocks

5.1 Implementing the code for a IO device

There are two kinds of IO blocks:

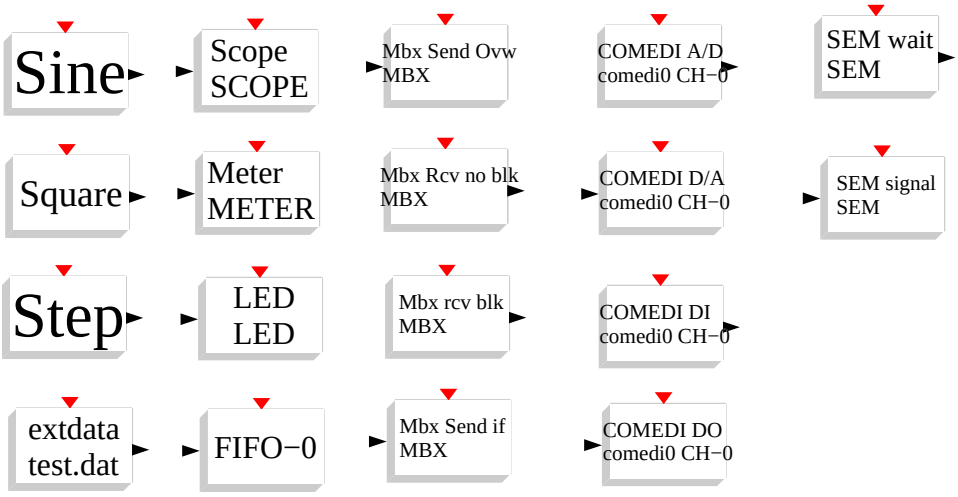

Figure 2: Scicos palette for RTAI

- Blocks which need RTAI resources (ex. RTAI APIs)
- Blocks which can be directly programmed using standard C functions.

The first case is normally programmend in two different files:

- A module for the RTAI library libsciblk.a, which can contain all the RTAI API calls
- A module for scicos which contains the code specific for scicos of this block

The new file must be created under "<rtaidir>/rtai-lab/scilab/devices". Different tools facilitate the implementation of the first file

- "template.c"
- "devtmpl.h"

Using the command

```
gen_dev <model>
```

a file named "<model>.c" will be created. This file contains all the needed functions to implement the driver as "input" and "output" driver. The utility "gen_dev" fills the file "devices.h" too.
The file "devstruct.h" contains the description of the structure used to store the block specific data.

```
typedef struct devStr{
  int nch;
  char IOName[20];
  char sName[20];
  char sParam[20];
  double dParam[5];
  int i1;
  long l1;
  long l2;
  void * ptr;
}devStr;
```

A structure for input (inpDevStr) and a structure for output (outDevStr) are provided in "rtmain.c". The differnet fields can be used to store temporary data related to the specific instance of the module.
Now the new file must be added to the GNUmakefile in order to be compiled and added to the "libsciblk.a" library.

- Go to the ".../rtai-lab/scilab/devices directory.
- Add the new file to the "libsciblk_a_SOURCES".
- Go to the RTAI main folder.
- Run the command "automake rtai-lab/scilab/devices/GNUmakefile". If there are problems with "automake" run "aclocal" before.

The last step is to run "make" and "make install" under "<rtaidir>/rtai-lab/scilab/devices".

5.2 Creating the Scicos block for sensor

A scicos file specific for a sensor block can be programmed starting from this example. Number 1 . . . 10 show the points of the program to be changed (see subsection 5.4).

```
//*****************************************************************************************
function [x,y,typ]=rtai_generic_inp(job,arg1,arg2)                                  // 1
//*****************************************************************************************
//
// Copyright roberto.bucher@supsi.ch
x=[];y=[];typ=[];
select job
case 'plot' then
  graphics=arg1.graphics; exprs=graphics.exprs;
//*****************************************************************************************
  name=exprs(1)(2);                                                                 // 2
//*****************************************************************************************
  standard_draw(arg1)
case 'getinputs' then
  [x,y,typ]=standard_inputs(arg1)
case 'getoutputs' then
  [x,y,typ]=standard_outputs(arg1)
case 'getorigin' then
  [x,y]=standard_origin(arg1)
case 'set' then
  x=arg1
  model=arg1.model;graphics=arg1.graphics;
  label=graphics.exprs;
  while %t do
//*****************************************************************************************
    [ok,op,name,lab]=..                                                             // 3
        getvalue('Set RTAI generic input block parameters',..
        ['output ports';
         'Identifier'],..
         list('vec',-1,'str',1),label(1))
//*****************************************************************************************

    if ~ok then break,end
    label(1)=lab
//*****************************************************************************************
    funam='i_generic_' + name;                                                      // 4
//*****************************************************************************************
    xx=[];ng=[];z=0;
    nx=0;nz=0;
    o=[];
```

```
    i=[];
    for nn = 1 : op
      o=[o,1];
    end
    o=int(o(:));nout=size(o,1);
    ci=1;nevin=1;
    co=[];nevout=0;
    funtyp=2004;
    depu=%t;
    dept=%f;
    dep_ut=[depu dept];

    [ok,tt]=getCode(funam)
    if ~ok then break,end
    [model,graphics,ok]=check_io(model,graphics,i,o,ci,co)
    if ok then
      model.sim=list(funam,funtyp)
      model.in=[]
      model.out=o
      model.evtin=ci
      model.evtout=[]
      model.state=[]
      model.dstate=0
//*****************************************************************************************
      model.rpar=[]                                                              // 5
//*****************************************************************************************
      model.ipar=[]
      model.firing=[]
      model.dep_ut=dep_ut
      model.nzcross=0
      label(2)=tt
      x.model=model
      graphics.exprs=label
      x.graphics=graphics
      break
    end
  end
case 'define' then
  out=1
  outsz = 1
//*****************************************************************************************
  name = 'SENS'                                                                  // 6
//*****************************************************************************************

  model=scicos_model()
  model.sim=list(' ',2004)
  model.in=[]
  model.out=outsz
  model.evtin=1
  model.evtout=[]
  model.state=[]
  model.dstate=[]
  model.rpar=[]
  model.ipar=[]
  model.blocktype='c'
  model.firing=[]
  model.dep_ut=[%t %f]
  model.nzcross=0

//*****************************************************************************************
```

```
  label=list([sci2exp(out),name],[])                                                    // 7

  gr_i=['xstringb(orig(1),orig(2),[''SENSOR'';name],sz(1),sz(2),''fill'');']
//*****************************************************************************************
  x=standard_define([3 2],model,label,gr_i)

end
endfunction

function [ok,tt]=getCode(funam)
   textmp=[
          '#ifndef MODEL'
          '#include <math.h>';
          '#include <stdlib.h>';
          '#include <scicos/scicos_block.h>';
          '#endif'
          '';
          'void '+funam+'(scicos_block *block,int flag)';
         ];
  textmp($+1)='{'
  textmp($+1)='#ifdef MODEL'
  textmp($+1)='int i;'
  textmp($+1)='double y[' + string(nout) + '];'
  textmp($+1)='double t = get_scicos_time();'
  textmp($+1)='  switch(flag) {'
  textmp($+1)='  case 4:'
//*****************************************************************************************
  textmp($+1)='    /* Initialisation */'                                                  // 8
//*****************************************************************************************
  textmp($+1)='    break;';
  textmp($+1)='  case 1:'
//*****************************************************************************************
  textmp($+1)='    /* input(y,t); */'                                                     // 9
//*****************************************************************************************
  textmp($+1)='    for (i=0;i<' + string(nout) + ';i++) block->outptr[i][0] = y[i];'
  textmp($+1)='    break;'
  textmp($+1)='  case 5:'
//*****************************************************************************************
  textmp($+1)='    /* end */'                                                            // 10
//*****************************************************************************************
  textmp($+1)='    break;'
  textmp($+1)='  }'
  textmp($+1)='#endif'
  textmp($+1)='}'

  tt=txttmp
  ok = %t
endfunction
```

5.3 Creating the Scicos block for actuator

A scicos file specific for an actuator block can be programmed starting from this example. Number 1 . . . 10 show the points of the program to be changed (see subsection 5.4).

```
//*****************************************************************************************
function [x,y,typ]=rtai_generic_out(job,arg1,arg2)                                      // 1
//*****************************************************************************************
//
// Copyright roberto.bucher@supsi.ch
x=[];y=[];typ=[];
```

```
select job
case 'plot' then
  graphics=arg1.graphics; exprs=graphics.exprs;
//*****************************************************************************************
  name=exprs(1)(2);                                                                  // 2
//*****************************************************************************************
  standard_draw(arg1)
case 'getinputs' then
  [x,y,typ]=standard_inputs(arg1)
case 'getoutputs' then
  [x,y,typ]=standard_outputs(arg1)
case 'getorigin' then
  [x,y]=standard_origin(arg1)
case 'set' then
  x=arg1
  model=arg1.model;graphics=arg1.graphics;
  label=graphics.exprs;
  while %t do
//*****************************************************************************************
    [ok,ip,name,lab]=..                                                              // 3
        getvalue('Set RTAI-mbx_send_if block parameters',..
        ['input ports';
         'Identifier'],..
         list('vec',-1,'str',1),label(1))
//*****************************************************************************************

    if ~ok then break,end
    label(1)=lab
//*****************************************************************************************
    funam='o_generic_' + name;                                                       // 4
//*****************************************************************************************
   xx=[];ng=[];z=0;
    nx=0;nz=0;
    o=[];
    i=[];
    for nn = 1 : ip
      i=[i,1];
    end
    i=int(i(:));nin=size(i,1);
    ci=1;nevin=1;
    co=[];nevout=0;
    funtyp=2004;
    depu=%t;
    dept=%f;
    dep_ut=[depu dept];

    [ok,tt]=getCode(funam)
    if ~ok then break,end
    [model,graphics,ok]=check_io(model,graphics,i,o,ci,co)
    if ok then
      model.sim=list(funam,funtyp)
      model.in=i
      model.out=[]
      model.evtin=ci
      model.evtout=[]
      model.state=[]
      model.dstate=0
//*****************************************************************************************
      model.rpar=[]                                                                  // 5
//*****************************************************************************************
```

```
      model.ipar=[]
      model.firing=[]
      model.dep_ut=dep_ut
      model.nzcross=0
      label(2)=tt
      x.model=model
      graphics.exprs=label
      x.graphics=graphics
      break
    end
  end
case 'define' then
  in=1
  insz = 1
//*********************************************************************************************
  name = 'ACT'                                                                          // 6
//*********************************************************************************************

  model=scicos_model()
  model.sim=list(' ',2004)
  model.in=insz
  model.out=[]
  model.evtin=1
  model.evtout=[]
  model.state=[]
  model.dstate=[]
  model.rpar=[]
  model.ipar=[]
  model.blocktype='d'
  model.firing=[]
  model.dep_ut=[%t %f]
  model.nzcross=0

//*********************************************************************************************
  label=list([sci2exp(in),name],[])                                                     // 7

  gr_i=['xstringb(orig(1),orig(2),[''ACTUATOR'';name],sz(1),sz(2),''fill'');']
//*********************************************************************************************
  x=standard_define([3 2],model,label,gr_i)

end
endfunction

function [ok,tt]=getCode(funam)
   textmp=[
          '#ifndef MODEL'
          '#include <math.h>';
          '#include <stdlib.h>';
          '#include <scicos/scicos_block.h>';
          '#endif'
          '';
          'void '+funam+'(scicos_block *block,int flag)';
         ];
  textmp($+1)='{'
  textmp($+1)='#ifdef MODEL'
  textmp($+1)='int i;'
  textmp($+1)='double u[' + string(nin) + '];'
  textmp($+1)='double t = get_scicos_time();'
  textmp($+1)='  switch(flag) {'
  textmp($+1)='  case 4:'
```

```
//*****************************************************************************************
  textmp($+1)='    /* init */'                                                        // 8
//*****************************************************************************************
  textmp($+1)='    break;';
  textmp($+1)='  case 2:'
  textmp($+1)='    for (i=0;i<' + string(nin) + ';i++) u[i]=block->inptr[i][0];'
//*****************************************************************************************
  textmp($+1)='    /* output(u,t); */'                                                // 9
//*****************************************************************************************
  textmp($+1)='    break;'
  textmp($+1)='  case 5:'
//*****************************************************************************************
  textmp($+1)='    /* end */'                                                         // 10
//*****************************************************************************************
  textmp($+1)='    break;'
  textmp($+1)='  }'
  textmp($+1)='#endif'
  textmp($+1)='}'

  tt=textmp
  ok=%t
endfunction
```

5.4 Fitting the code

1. Here the name of the function must be set.
2. The user has the possibility to extract some fields which can be used by the "plot" function, in order to display some values on the scicos block here.
3. This is the dialog box needed to get all the parameters and info of the block.
4. A name of the generated C-function is generated here.
5. Some block parameters can be inserted in the "rpar" fields. These parameters can be modified by the "xrtailab" application.
6. All the parameters of the dialog box (see point 3) should be initialized here.
7. The block default values and the block look are set here.
8. The initialization code of the block must be programmed here.
9. The input respectively output code of the block must be programmed here.
10. The termination code of the block must be programmed here.

The new blocks must be added to the "Makefile" in the "/macros/RTAI" directory. The user must now simply run "make" to complete the procedure.

6 Using COMEDI drivers

In order to use COMEDI drivers with the RTAI-Lab environment the following modules should be installed:

- rtai_hal
- rtai_lxrt
- rtai_fifos
- rtai_sem

- rtai_mbx
- rtai_msg
- rtai_netrpc
- rtai_shm
- rtai_comedi
- comedi
- kcomedilib

At this point the user must load the COMEDI specific modules and perform the "comedi_config" command.