

CONNECTING ECOSIMPRO AND SYSQUAKE IN A NONLINEAR INTERACTIVE CONTROL TOOL

Sebastián Dormido Bencomo

Dpto. Informática y Automática, Facultad de Ciencias

UNED, Fac. de Ciencias, Dpto. de Informática y Automática. Avda. Senda del Rey 9, 28040 Madrid, Spain

e-mail: sdormido@dia.uned.es

Ramón Perez Vara and Pedro Cobas

Empresarios Agrupados

Magallanes-3, 28015 Madrid, Spain

e-mail: pce@ecosimpro.com, rpv@empre.es

Abstract

SysQuake is a tool which has been designed for simulating and displaying scientific data. With its innovative interactive graphics concept and a base language which is essentially a Matlab clone, SysQuake offers very powerful, simple base primitives for resolving complex mathematical problems in which interaction plays a key role. This is precisely the kind of situation which arises with control problems in which representations have to be correlated in the frequency and temporal domains. This work describes how to connect SysQuake with EcosimPro in a concrete example which demonstrates how interaction can be of tremendous use in hybrid system object oriented causal modelling languages.

Key Words: Control education, EcosimPro, interaction, SysQuake.

1 INTRODUCTION

Ideas of automatic control are very rich in visual content which can be represented intuitively and geometrically. These visual contents can be employed to present tasks and manage concepts and methods, and to manipulate them in the resolution of problems.

The basic ideas of automatic control often arise from very specific visual situations and all the control experts recognise the great advantage to be gained from capitalising on them when they have to manipulate the corresponding abstract objects.

Giving explicit attention in this way to potential specific representations to explain abstract relationships which are of interest is what we term visual displays in control.

Our perception is basically visual and it should therefore not surprise us that visual aids are ever present in our work. Very often we use symbolic processes, visual diagrams and other forms of imaginative processes which enable us to acquire what could be called a certain intuition of the abstract form.

Displays seem to be something profoundly natural both in the origin of automatic control and in the discovery of new relationships between mathematical objects, and of course in the transmission and communication of our knowledge of control.

Students will probably find it much more difficult to precisely assert and assimilate these intuitive aspects because they are very often found at the least conscious level in the activity of a specialist (Dormido, 2002).

Based on these general considerations, the computer can be regarded as a tool which enables us to interactively display and manipulate objects which are peculiar to automatic control. The final objective is to facilitate the understanding of the concepts we want to transmit to our students.

Systems are traditionally designed following an iterative process. The specifications of a problem are not normally used to calculate the value of system parameters because there is no explicit formula which directly relates them. This is why each iteration is divided into two stages. The first, often called *synthesis*, consists in calculating the unknown parameters of the system based on a group of design variables (which are related to the specifications). During the second stage, called *analysis*, the system behaviour is evaluated and compared with the specifications. If they do not tally, the design variables are modified and the iteration process is repeated. The two stages can, however, be combined so that modification of the parameters produces an immediate effect. This makes the design process

really dynamic and the student can perceive the gradient of change in the behaviour criterion for the elements being manipulated. This interactive system makes it much easier to determine [los compromisos que se pueden alcanzar???] the scope of difficulty that can be managed.

Over the past few years many tools have been developed for use in education in control. Many interesting ideas and concepts were implemented by Professor Åström's group at Lund Institute of Technology. In this context it is worth noting the concepts of *dynamic tables* and *virtual interactive systems* introduced by Wittenmark *et al*, 1998. The main objective of these tools is to achieve far more active participation of students in automatic control courses.

In essence, a dynamic table is a collection of graphic windows which are manipulated simply with the use of the mouse. Students are not required to write any code sentence or learn anything beyond the focal objective of their course. If students change any active element in the graphic windows, automatically a new process is started and the results are displayed. In this way they can appreciate how their modifications affect the results obtained.

These interactive tools try to "demythologise" abstract mathematical concepts by displaying examples selected ad hoc. A new generation of software packages has created an interesting alternative for the interactive learning of automatic control (Garcia and Heck, 1999). The operating principle is based on objects which enable direct graphic manipulation. With this manipulation the objects are updated immediately so the relationship between objects is maintained at all times. *Ictools* and *CCSdemo* (Johansson *et al*, 1998; Wittenmark *et al*, 1998), developed in the Department of Automatic Control at the University of Lund and *SysQuake* developed in the Institute for Automatic Control of the Federal Polytechnic School of Lausanne, (Pyguet, 1999).

With this philosophy, an interactive tool has been developed to explain the basic concepts of nonlinear control (Dormido *et al*, 2002) using SysQuake as the working tool. The problem with SysQuake is that it does not automatically incorporate the detection of discrete events planned in a state, and this greatly complicates the calculation of paths / planes in the state space. On the other hand, this detection is very natural in causal modelling languages like that used by EcosimPro (EL). From this perspective, we propose connecting SysQuake and EcosimPro to simplify the interactive solution of this type of problem.

The problem is briefly described in section 2. In section 3 we explain how a model built with EcosimPro is connected to SysQuake. Section 4 briefly describes the solution adopted when the nonlinear control described in section 2 is applied. Finally, the conclusions of this work are given in section 5.

2. DESCRIPTION OF NONLINEAR CONTROL

The systems that can be studied are of the kind illustrated in Figure 1. The linear element is any user-defined transfer function (future versions of the tool are expected to incorporate pure time lags).

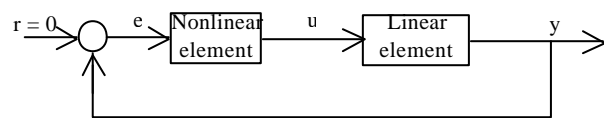


Figure 1: System Structure

The nonlinear element is of the general kind depicted in Figure 2 which covers a large variety of nonlinearities which are linear in sections with odd symmetry, and which includes the majority of the classical multivaluated nonlinearities (Sridhar, 1960). With it we can describe the following nonlinearities, to name but a few: [zona muerta] neutral zone, saturation, neutral zone + saturation, relay and relay with hysteresis.

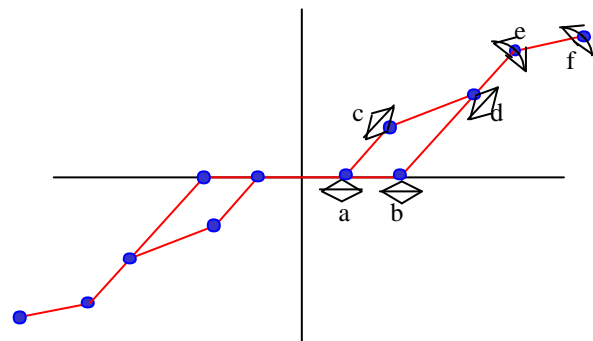


Figure 2: Generic Nonlinearity

The nonlinearity is configured interactively: using the mouse, the user only has to move the active points (represented by small circle) in the "Nonlinear element" window of the tool. The direction of the arrows indicates the type of displacement that can be exercised on each of the points.

Figure 3 illustrates the other windows incorporated into the tool. These windows are: "Parameters", "Transfer function", "Input to nonlinear element", "Real part and imaginary part describing the function", "Nyquist curve and critical locus", "Phase plane", "Input" and "Output". For a detailed analysis of the application refer to Dormido *et al*, 2002.

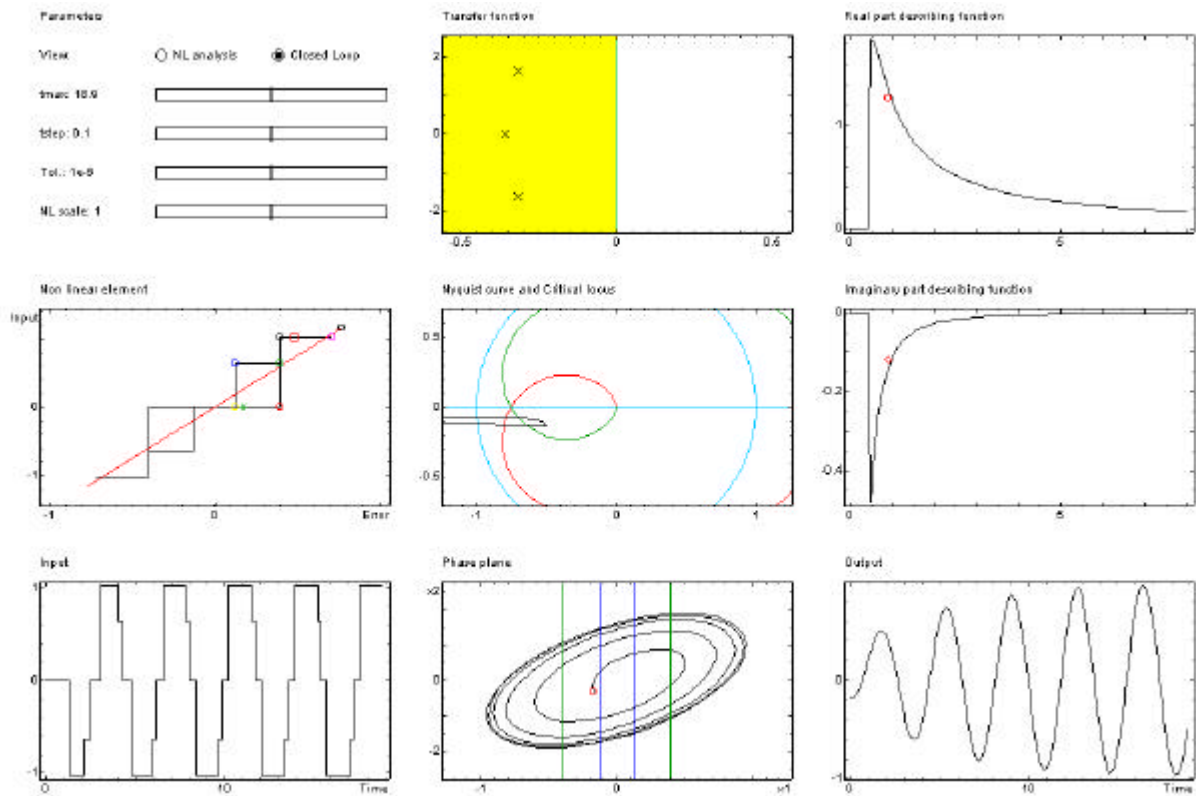


Figure 3 Multiple View of the Windows incorporated into the Tool

3. CONNECTING AN ECOSIMPRO MODEL WITH SYSQUAKE

It is easy to connect a model generated with EcosimPro to SysQuake; all we have to do is create a *dll interface which calls the EcosimPro model*. As usual in these cases, the difficulty is doing it for the first time because the programming framework will be identical or very similar for successive runs. Sometime in the future it will be possible to evaluate the automatic generation of the dll, but for the time being the programming is manual.

We have to start by saying that the following is required to create this interface:

- Knowledge of MS Visual C++. Specifically, the creation and management of dynamic libraries and programming in C++
- Familiarity with the SysQuake manual as regards connection to external routines
- Familiarity with the manual for connecting EcosimPro to SysQuake [Cobas, 2002]

Figure 4 illustrates the connection process:

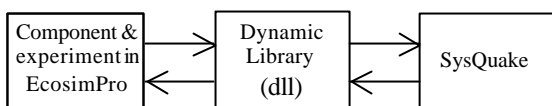


Figure 4: EcosimPro-SysQuake Connection Diagram

First of all the system has to be modelled with EcosimPro and it has to be thoroughly tested. This will guarantee that the model built is robust and is ready to work with any input data. For models which are not trivial this can be an arduous and difficult task. It must be taken into account that one of SysQuake's strong points is that we can make the user change any input datum interactively and that is why it must be guaranteed that the model will respond adequately.

Once the EcosimPro model is working correctly for any type of input we can connect it to SysQuake by carrying out the following steps:

- Determine which are the input data to and output data from the model
- Create an experiment which integrates into the model a time, which could be given by SysQuake
- Create some arrays in the experiment which will store the variables we want to return to SysQuake
- Test this experiment thoroughly
- Create a dll with the generated code which links this model to SysQuake
- If appropriate (advanced handling of SysQuake), program a code in SysQuake to connect it to the external function of EcosimPro (dll) which adequately defines the inputs to and outputs from the function. If no code is created, it is always

possible to directly call the created function from SysQuake (see example further on)

- Start SysQuake and check that the dll loads the developed dll function
- Execute SysQuake changing the variables and checking that the results which are drawn in SysQuake are correct

A very brief explanation of these steps is given below (for details, refer to Cobas, 2002) for a trivial component in EcosimPro which resolves the equation “ $x = \sin(\text{TIME}) + d1$ ”, where x will be the variable to be calculated, TIME the actual integration time and $d1$ a known datum which the user will be able to send from SysQuake.

First of all we model the component in EcosimPro:

```

COMPONENT pruSysQuake
      DATA
      REAL d1 = 1.9
      DECLS
      REAL x
CONTINUOUS
      x = sin(TIME) + d1
END COMPONENT

```

Then we write an experiment which integrates a time into this model and stores the values of TIME and x in an array:

```

EXPERIMENT expl ON
pruSysQuake.default
DECLS
  CONST INTEGER maxDatos= 400
  REAL mtime[maxDatos]
  REAL mx[maxDatos]
  INTEGER i
BODY
  TIME= 0 - CINT
  i = 1
WHILE (INTEG_CINT() !=INTEG_END)
  mtime[i]= TIME --rellena con t
  mx[i] = x      --rellena con x
  i = i + 1
END WHILE
END EXPERIMENT

```

The input datum will be “ $d1$ ” and the output arrays will be “ $mtime$ ” and “ mx ”. In the example we will integrate a fixed interval which is initialised in the dll. In a more sophisticated case the input data could be TSTOP and CINT.

3.1 CREATING THE DLL

The dll is created by generating a project in Visual C++ for which we create a new project such as “Win32 Dynamic-link library”. This project will include the classes generated by EcosimPro and

others used by SysQuake (see details in Cobas, 2002). The main function of the dll must have a structure of the following type:

```

static lme_int32
ecosimSeno(lme_ref lme,
lme_int32 nargin, lme_int32
nargout)
{
...
}

```

The important arguments are `nargin` (number of input arguments) and `nargout` (number of output arguments). Both must be initialised in this function. In our case `nargin = 1` (datum “ $d1$ ”) and `nargout = 2` (arrays “ $mtime$ ” and “ mx ”).

All that is required to connect to the EcosimPro model is to request the class of the experiment and call the method `initEcosim`. For example:

```

pruSysQuake _default_expl
modelo;
initEcosim( &modelo );

```

We now have an EcosimPro model in the dll. From here, three steps will be carried out:

1. The SysQuake input matrices will be read and passed to the EcosimPro model
2. The model will be executed
3. The results will be read and transmitted to SysQuake in matrix format

3.2 READING THE INPUT ARRAYS

In order to read $d1$, it has to be passed as a 1 dimension matrix. To do this the function `LMECB_GetMatrix()` must be used, for example:

```

lme_int32 status=
LMECB_GetMatrix(narg, &nf, &nc,
&D, 0);
if (!status)
return 0;
modelo.setValueReal("d1", D[0]);

```

The last line assigns the value that `D[0]` contains in the “ $d1$ ” datum of the model. There is another way of doing it which is more awkward but quicker. The internal address of any variable of the EcosimPro model can be obtained as follows:

```

double* d1= (double*)
modelo.getVarAddress("d1");

```

From now on it is as though the variable `d1` were pointing to the `d1` of the EcosimPro model. This enables the following, for example:

```
*dl = D[0];
```

And this would be exactly the same as what we did with the first method.

3.3 ASSIGNING THE OUTPUT ARRAYS

SysQuake has to be informed of the addresses the output arrays have in the memory. There will be two arrays belonging to SysQuake which will subsequently be filled with the results.

3.4 PROGRAMMING THE EXECUTION OF SIMULATION

To run the simulation, the initial and final time and the communication interval can be assigned easily from the dll. They could also be passed from SysQuake, but for simplification in this case we establish the fixed simulation times and integration method:

```
modelo.TIME= 0;
modelo.TSTOP= 1;
modelo.CINT= 0.2
modelo.IMETHOD= RK4;
```

Afterwards, the experiment is executed:

```
modelo.runExperiment(); //
ejecuta el experimento de Ecosim
```

3.5 READING THE OUTPUT ARRAYS

Finally, the results of EcosimPro will have to be read and passed to SysQuake.

```
modelo.getArray1D("mtime", Time,
nDatos); // Lee Time
modelo.getArray1D("mx", X,
nDatos); // Lee X
```

This call fills the arrays Time and X with the values read in "mtime" and "mx". SysQuake now has the results.

3.6 INITIALISING THE EXTERNAL FUNCTION

Apart from the function "ecosimSeno", a function called "InstallFn" which loads the dll in automatic mode and an lme_fn static structure which defines the name and the number of arguments have to be created. The objective is for SysQuake to automatically detect that there is a new function to be loaded.

3.7 EXECUTING SYSQUAKE

Once the dll is generated it will be placed in the SysQuake LMEEExt directory and it must be automatically loaded when the SysQuake program is executed (a message should appear to confirm this). From now on the ecosimSeno() function will be available in SysQuake and it can be called with different dl values and we would obtain two matrices: one with the TIME values and the other with the X values. For example, by passing the function:

```
ecosimSeno([1.9])
```

we obtain

```
mtime[0, 0.2, 0.4, 0.6, 0.8, 1.0]
mx [1.9, 2.0986, 2.2894,
2.4646, 2.6173, 2.7414 ]
```

This function can be used from the SysQuake code as if it were yet another function of the SysQuake library.

4. CONNECTING AN ECOSIMPRO MODEL WITH SYSQUAKE

From the point of view of calculating the paths / planes of application of nonlinear control described in section 2, the problem is a chain of linear path sections between every two consecutive "cross points" because the nonlinear element can be modified interactively by the user. Since SysQuake is not equipped with automatic mechanisms for detecting the cross points, this is a task which has to be developed within a previously defined tolerance margin. The problem is particularly difficult from a numeric point of view because for certain linearities the path / plane can present "sliding mode" elements.

An interesting alternative which has been implemented in this work is that of combining the interactive capabilities of SysQuake with the possibility of using EcosimPro's built-in solvers which automatically detect the cross points through nonlinearity without requiring any action on the part of the user. In essence, SysQuake interactively modifies the following elements:

- 1- Nonlinear element, which is described by the coordinates of the points that define it

```
[xa xb xc yc xd yd xe ye xf yf]
```

- 2- Linear transfer function, described by the associated model in the state space

```
[A B C D]
```

3- Initial condition of the state vector x_0

4- Dynamic simulation parameters

```
[tmax tstep metodo_integracion]
```

For its part, EcosimPro returns the moments of time vector t , the state vector x and the output values vector at these moments y .

5. CONCLUSIONS

This work has demonstrated how SysQuake and EcosimPro can be connected in an interactive application where the determination of events in a state are detected by the EcosimPro solver. This approach greatly simplifies program writing and facilitates debugging. The ideas presented can be easily transferred to analogue situations and the connection procedure of both tools can be even further automated. One aspect which this work has brought to light is the possibility of increasing EcosimPro's potential by equipping its language for describing "experiments" with primitives which facilitate the development of interactive applications. This task does not present conceptual problems when the interactive variables which are modified do not change the "index of the problem".

References

- Cobas, P.: (2002). "Manual de conexión de un modelo EcosimPro a SysQuake", EA International. Nov-2002
- Dormido, S. (2002) Control Learning: Present and Future. Plenary Lecture. 15th Triennial World Congress of IFAC, Barcelona, Spain
- S. Dormido, F. Gordillo, S. Dormido-Canto, J. Aracil (2002). An interactive tool for introductory nonlinear control systems education. 15th Triennial World Congress of IFAC, Barcelona, Spain.
- Garcia, R.C.; Heck, B.S.: (1999). "Enhancing classical controls education via interactive GUI design", *IEEE Control Systems Magazine*, **19**, No. 3, p. 77-82.
- Johansson, M.; Gäfvert, M.; Åström, K.J.: (1998). "Interactive tools for education in automatic control", *IEEE Control Systems Magazine*, **18**, No. 3, p. 33-40.
- Piguet, Y.: (1999). "SysQuake: User Manual", Calerga.
- Wittenmark, B.; Häglund, H.; Johansson, M.: (1998). "Dynamic pictures and interactive learning", *IEEE Control Systems Magazine*, **18**, No. 3, p. 26-32.