DrControl — An Interactive Course Material for Teaching Control Engineering

Mohsen Torabzadeh-Tari, Martin Sjölund, Adrian Pop, Peter Fritzson

PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
{mohsen.torabzadeh-tari, martin.sjolund}@liu.se
{adrian.pop, peter.fritzson}@liu.se

Abstract

In this paper we present an interactive course material called DrControl for teaching control theory concepts mixed together with exercises and example models in Modelica.

The active electronic notebook, OMNotebook, is the basis for the course material. This can be an alternative or complement compared to the traditional teaching method with lecturing and reading textbooks. Experience shows that using such an electronic book will lead to more engagement from the students. OMNotebook can contain interactive technical computations and text, as well as graphics. Hence it is a suitable tool for teaching, experimentation, simulation, scripting, model documentation, storage, etc.

Keywords: DrControl, DrModelica, modeling, simulation, OMNotebook, teaching, interactive, Control

1 Introduction

In this paper we introduce an electronic interactive course material called DrControl and its use for teaching control theory together with control applications in Modelica [1] [2]. It is developed in and uses the OMNotebook [5] active electronic book software together with OpenModelica for modeling and simulation.

This kind of interactive courses based on electronic books allows experimentation and dynamic simulation as well as execution of computer programs.

Traditional teaching methods with lecturing and reading a textbook are often too passive and does not engage the student. Active notebooks, however, facilitates the learning process, e.g. by running programs and exercises within the book, and mixing lecturing with exercises and with reading in the interactive book.

Electronic notebooks created using OMNotebook can contain program code, text, links, pictures, video, virtual and scientific visualizations, and makes it is possible to integrate teaching material in sciences such as physics, human biology [3], mathematics, computer science, etc.

1.1 Structure of the Paper

Section 2 presents the OMNotebook tool, whereas Section 3 describes the teaching goals and contents of the DrControl electronic book. Section 4 briefly mentions applications in teaching modeling and programming languages, whereas Section 5 presents future work and Section 6 gives the conclusions..

2 OMNotebook – An Active Electronic Notebook

The OpenModelica Notebook editor, OMNotebook, provides an active electronic notebook including an editor. The notebook it is not just a passive textbook or html page, it is active in the sense that models inside the book can be changed and executed.

This functionality allows the usage of interactive hierarchical text documents where the underlying chapters and sections can be represented and edited. OMNotebook supports functionality for Modelica model simulation [1] [2], text, images and interactive linking between those. Furthermore, via the external interface, program is other languages can be evaluated. One example is OMScheme (Section 4.2) for teaching the Scheme programming language.

The hierarchical structure of traditional documents, e.g. books and reports, can also be applied to the notebook which means basically that the book is divided into sections, subsections, paragraphs, etc. This makes the navigation in the book sections easier.

2.1 DrControl

Application of OMNotebook in control theory with the DrControl course material aims at reinforcing the understanding through practical applications with hands-on experience. The students gain insight into the dynamic phenomena of a system. Also, the problem-solving process can be built into the material thus letting the students explore the content at his or hers own convenience.
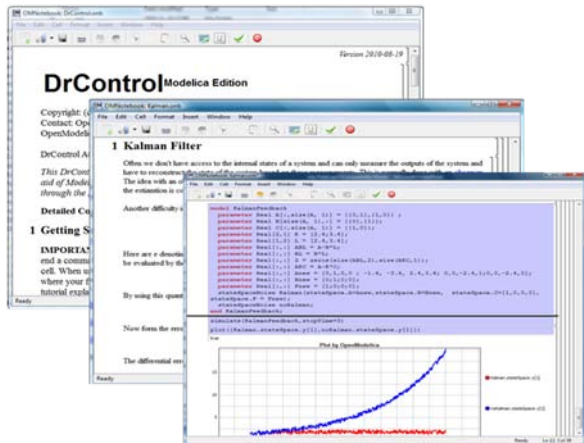

Figure 1. DrControl for teaching control theory concepts.

3 Content and Learning Goals of DrControl

One important factor in modeling and simulation is the availability of the source code, documentation of the source code as well as the result of the simulation in the same document. This is important because the problem solving process is an iterative process that requires modification of the original mathematical model and/or the software implementation and verification of the simulation result against the model.

The front-page of DrControl shown in Figure 2 resembles a linked table of content that can be used as a navigation center. The content list contains topics like:

- Getting started
- The control problem in ordinary life
- Feedback loop, see Section 3.1
- Mathematical modeling, see Section 3.2
- Transfer function, see Section 3.3
- Stability
- Example of controlling a DC-motor
- Feedforward compensation
- State-space form, see Section 3.4

- State observation, see Section 3.5
- Closed loop control system.
- Reconstructed systems, see Section 3.5
- Linear quadratic optimization, see Section 3.6
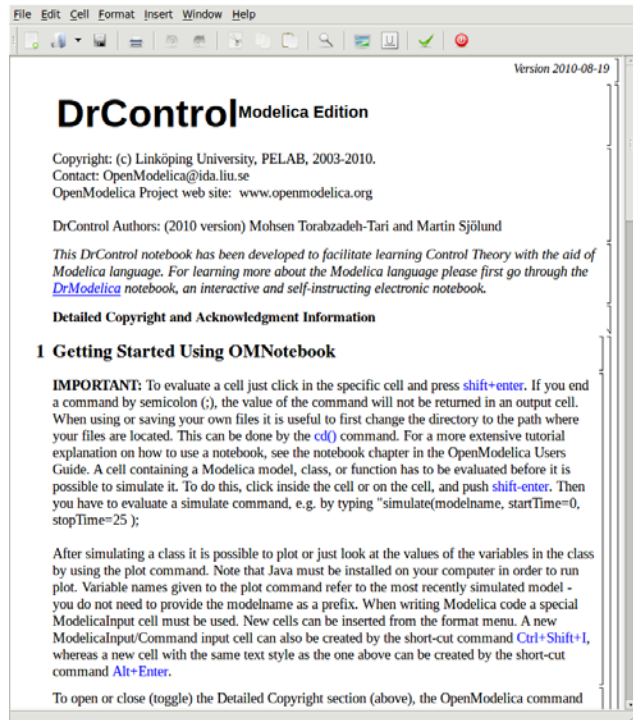- Linearization, see Section 3.7

Figure 2. The starting page of the DrControl tutoring system using OMNotebook.

Each entry in this list leads to a new notebook page where either the theory is explained with Modelica examples or an exercise with a solution is provided to certify the background theory, see [7] for more information and down-load of DrControl.

3.1 Feedback Loop

One of the basic concepts of control theory is using feedback loops either for neutralizing the disturbances from the surroundings or a desire for a smoother output.

In Figure 5 a simple car model is illustrated where the car velocity on a road is controlled, first with an open loop control then compared to a closed loop system with a feedback loop. The car has a mass m, velocity y, and aerodynamic coefficient α. The θ is the road slope, which in this case can be regarded as noise.
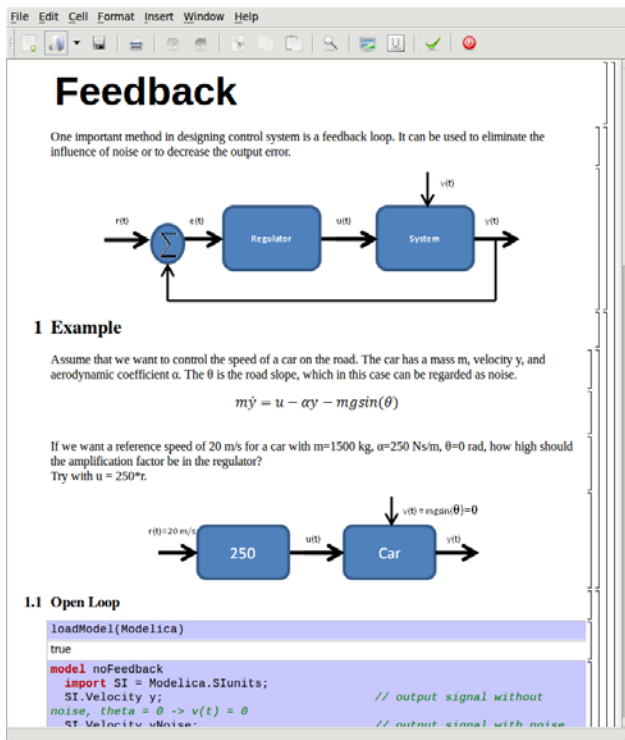
Figure 3. Feedback loop.

Lets look at the Modelica model for the open loop controlled car:

$$m\dot{y} = u - \alpha y - mg sin(\theta)$$

```
model NoFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y         "No noise";
  SI.Velocity yNoise    "With noise";
  parameter SI.Mass m = 1500;
  parameter Real alpha = 200;
  parameter SI. ngle theta = 5*3.14/180;
  parameter SI.Acceleration g = 9.82;
  SI.Force u;
  SI.Velocity r = 20 "Reference signal";
equation
  m*der(y)=u - alpha*y;
  m*der(yNoise)= u - alpha*yNoise -
     m*g*sin(theta);
  u = 250A*r;
end NoFeedback;
```

By applying a road slope angle different that zero then the car velocity is influenced which can be regarded as noise in this model. The output signal in Figure 3 is stable but an overshoot can be observed compared to the reference signal. Naturally the overshoot is not desired and the student will in the next exercise learn how to get rid of this undesired behavior of the system.
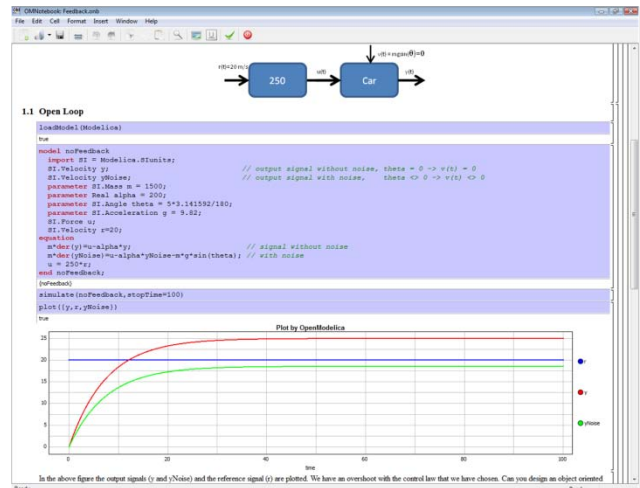

Figure 4. Open loop control example.

The closed car model with a proportional regulator is shown below:

$$u = K * (r - y)$$

```
model WithFeedback
  import SI = Modelica.SIunits;
  SI.Velocity y         "Output, No noise";
  SI.Velocity yNoise  "Output With noise";
  parameter SI.Mass m = 1500;
  parameter Real alpha = 250;
  parameter SI.Angle theta = 5*3.14/180;
  parameter SI.Acceleration g = 9.82;
  SI.Force u;
  SI.Force uNoise;
  SI.Velocity r = 20    "Reference signal";
equation
  m*der(y) = u - alpha*y;
  m*der(yNoise) = uNoise - alpha*yNois -
     m*g*sin(theta);
  u = 5000*(r - y);
  uNoise = 5000*(r - yNoise);
end WithFeedback;
```

By using the information about the current level of the output signal and re-tune the regulator the output quantity can be controlled towards the reference signal smoothly and without an overshoot, as shown in Figure 5.

In the above simple example the flat modeling approach was adopted since it was the fastest one to quickly obtain a working model. However, one could use the object oriented approach and encapsulate the car and regulator models in separate classes with the Modelica connector mechanism in between.
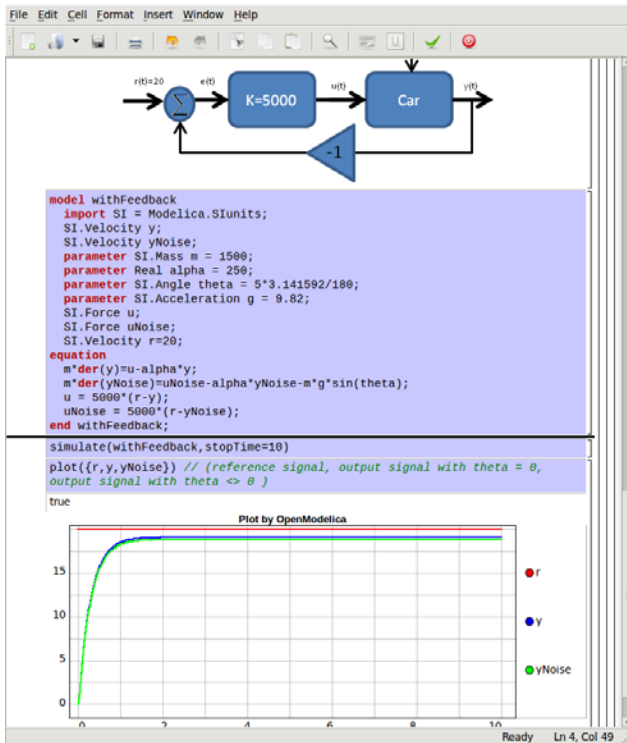
Figure 5. Closed loop control example.

3.2 Mathematical Modeling

In most systems the relation between the inputs and outputs can be described by a linear differential equation. Tearing apart the solution of the differential equation into homogenous and particular parts is an important technique taught to the students in engineering courses, also illustrated in Figure 6.

$$\frac{d^n y}{dt^n} + a_1 \frac{d^{n-1} y}{dt^{n-1}} + \ldots + a_n y$$
$$= b_0 \frac{d^m u}{dt^m} + \cdots + b_{m-1} \frac{du}{dt} + b_m u$$

Now let us examine a second order system:

$$\ddot{y} + a_1 \dot{y} + a_2 y = 1$$

```
model NegRoots
  Real y;
  Real der_y;
  parameter Real a1 = 3;
  parameter Real a2 = 2;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end NegRoots;
```

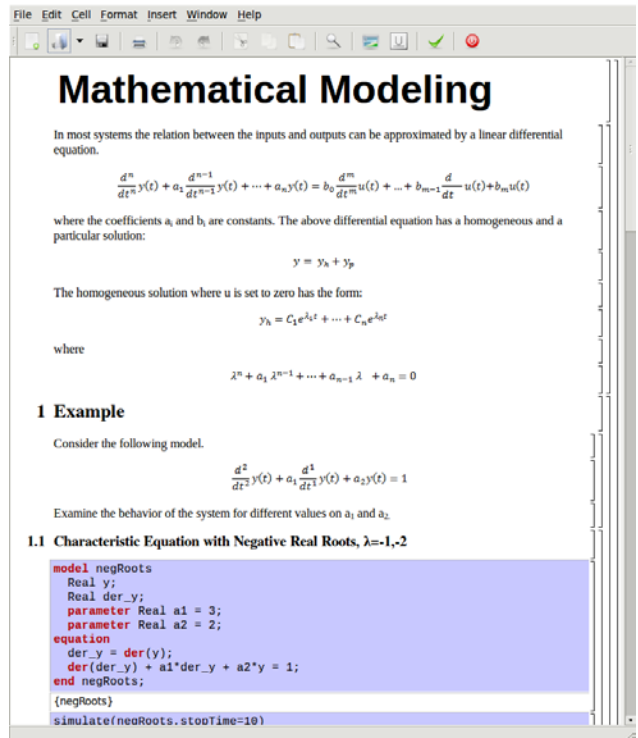Choosing different values for a_1 and a_2 leads to different behavior as shown in Figure 7 and Figure 8.

Figure 6. Mathematical modeling.

In the first example the values of a_1 and a_2 are chosen in such way that the characteristic equation has negative real roots and thereby a stable output response, see Figure 7.
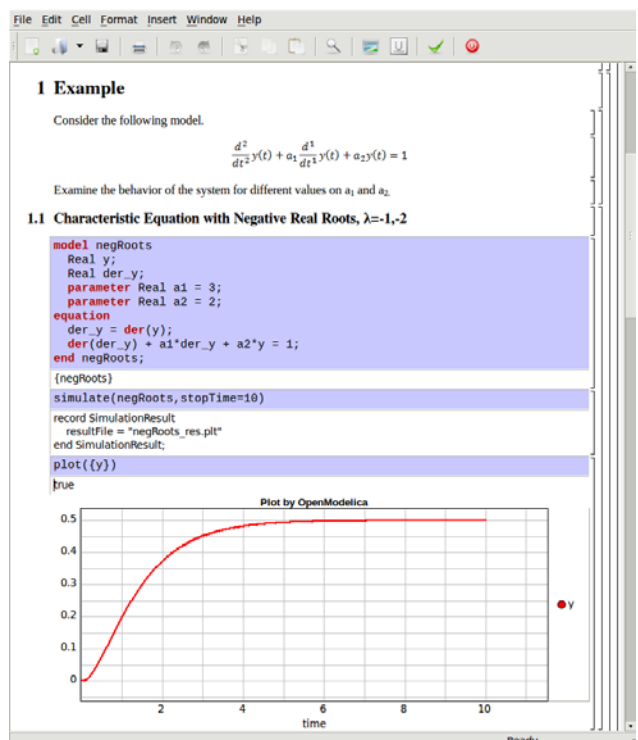

Figure 7. Characteristic eq. with real negative roots.

The importance of the sign of the roots in the characteristic equation is illustrated in Figure 7and Figure 8, e.g. a stable system with negative real roots and an unstable system with positive imaginary roots resulting in oscillations.

```
model NegRoots
  Real y;
  Real der_y;
  parameter Real a1 = -2;
  parameter Real a2 = 10;
equation
  der_y = der(y);
  der(der_y) + a1*der_y + a2*y = 1;
end NegRoots;
```
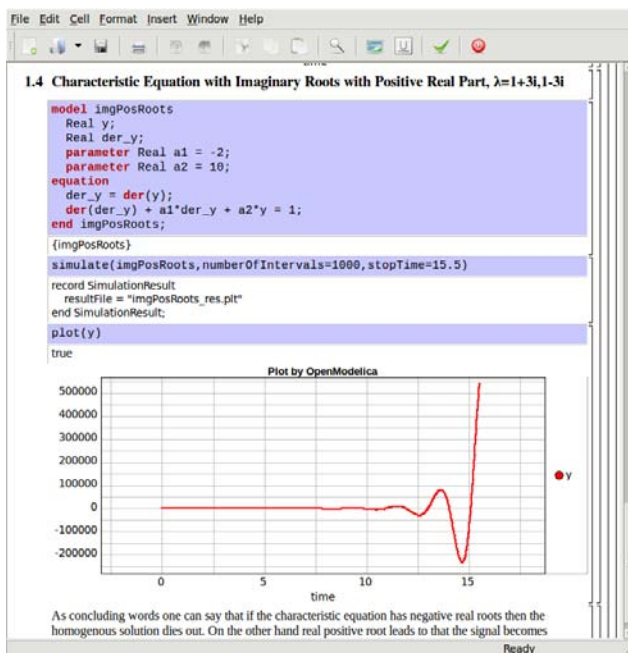
Figure 8. Characteristic eq. with positive imaginary roots.

3.3 Transfer Function

Students also get familiar with how a transfer function, polynomial fraction of the Laplace transform of output over the input, is derived and how it can be used to study the system behavior, see Figure 9 and Figure 10.

The poles of the transfer function are the roots of the denominator which is the same as the roots to the characteristic equation. The zeros are the roots to the numerator of the transfer function. The inverse Laplace transform of G(s) is called the weight function and is the impulse response of the system.

$$Y(s) = G(s)U(s)$$

Lets now look at a simplified first order model of a tank system:

$$G(s) = \frac{\frac{1}{A}}{s + \frac{1}{T}}$$

```
model Tank
  import Modelica.Blocks.Continuous.*;
  TransferFunction G(b = {1/A}, a =
    {1,1/T});
  TransferFunction GStep(b = {1/A},a =
    {1,1/T});
  parameter Real T = 15 "Time constant";
  parameter Real A = 5;
  Real uStep = if (time > 0 or time<0)
    then 1 else 0 "step function";
initial equation
  G.y = 1/A;
equation
  G.u= if time > 0 then 0 else 1e6;
  GStep.u = uStep;
end Tank;
```
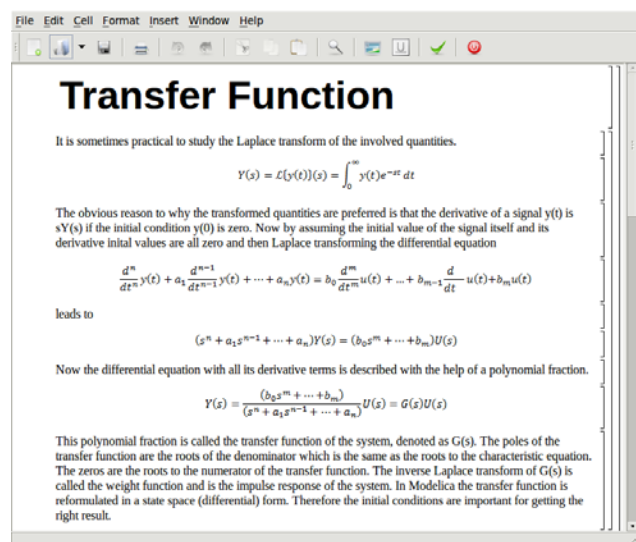
Figure 9. Transfer function derivation.

For analysis of a simple tank model the step and pulse responses of this system are illustrated in Figure 10. In Modelica the transfer function is reformulated in a state space (differential) form. Therefore the initial conditions are important for getting the right result.

The inverse Laplace transform of G(s) is called the weight function and is the impulse response of the system. In Modelica the transfer function is reformulated in a state space (differential) form. Therefore the initial conditions are important for getting the right result.
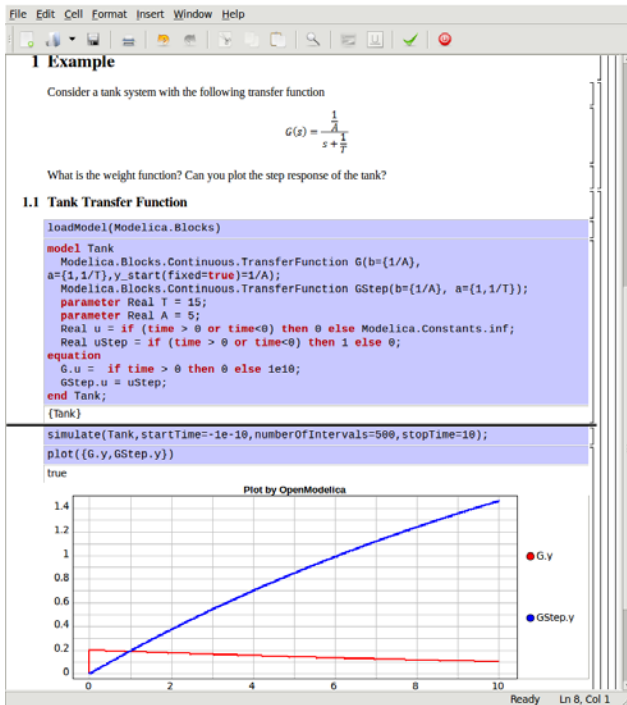
Figure 10. Step and pulse (weight function) response.

3.4 State-space Formulation

The state of a system is the amount of information needed for determining the future output of the system if the future inputs are known.

The state space form for continuous-time dependent systems can be expressed as a system of first order differential equations. We can reformulate the below second order differential equation

$$\ddot{y} + a_1\dot{y} + a_2 y = bu$$

by introducing new auxiliary variables

$$\begin{cases} x_1 = y \\ x_2 = \dot{y} \end{cases}$$

the differential equation can be re-written in a state-space form:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -a_2 & -a_1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 0 \\ b \end{pmatrix}u$$

Depending of the modeled system and the type of analysis one would like to perform there could be a desire to shift from the state space formulation to transfer function representation or vice versa.
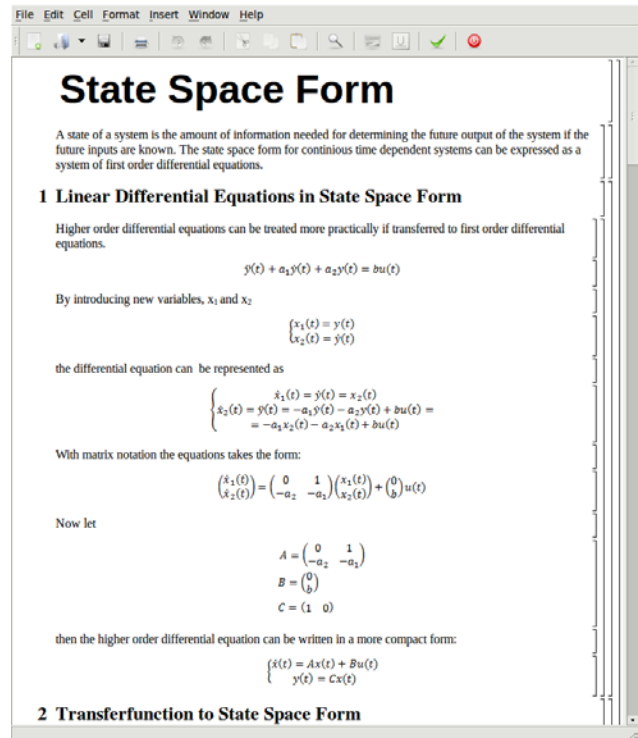

Figure 11. Linear state-space form.

In Figure 12 a second order system is modeled, both with the aid of pure differential equation and also with the transformation to the transfer function representation.

What is important to highlight here is that the two models show different results making the student aware of setting the initial data correctly.

```
model StateSpaceHD
  Modelica.Blocks.Continuous.StateSpace
    stateSpace(A=[-2,1; -3,0],B=[-3;5]
               ,C=[1,0],D=[2]);
  Modelica.Blocks.Sources.Step
    step(height=1.0);
equation
  connect(step.y, stateSpace.u[1]);
end StateSpaceHD;

model DiffEqHD
  Real u = 1;
  Real y;
  Real uprim = der(u);
  Real z = der(y);
equation
  der(z)+2*z+3*y = 2*der(uprim)+uprim+u;
end DiffEqHD;
```
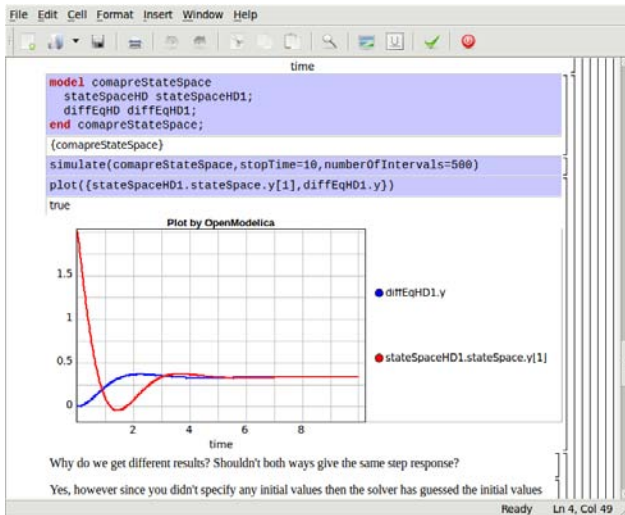
Figure 12. State-apace form vs. differential equation modeling.

3.5 Observers and Reconstructed systems

Often we do not have access to the internal states of a system and can only measure the outputs of the system and have to reconstruct the state of the system based on these measurements. This is normally done with an observer, e.g. Kalman filter, see Figure 13 and Figure 14.
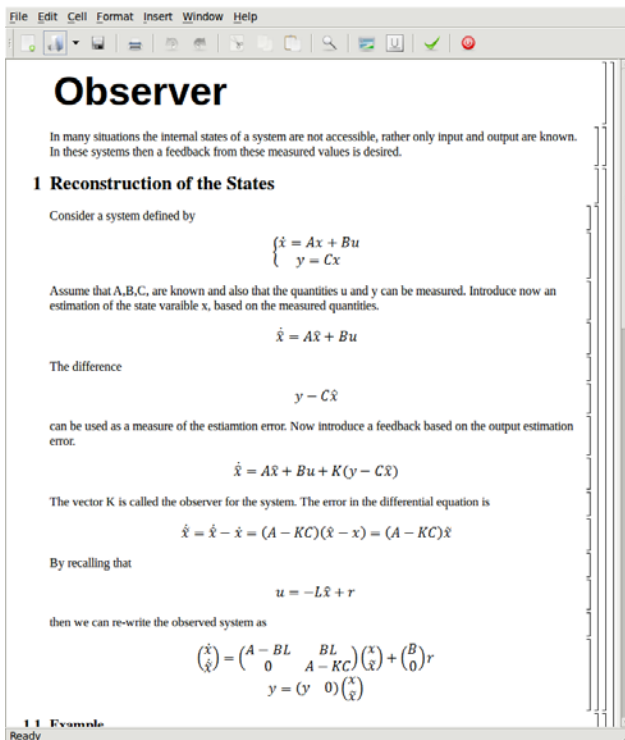

Figure 13. Observer.

Consider the second order model from section 3.4

$$\begin{cases} \dot{x} = Ax + Bu \\ \quad y = Cx \end{cases}$$

Introduce now an estimation of the state variable x:

$$\dot{\hat{x}} = A\hat{x} + Bu$$

The difference

$$y - C\hat{x}$$

can be used as a measure of the error in this estimation. With the feedback loop

$$u = -L\hat{x} + Br$$

the observed system can be re-written as:

$$\begin{pmatrix} \dot{x} \\ \dot{\hat{x}} \end{pmatrix} = \begin{pmatrix} A - BL & BL \\ 0 & A - KC \end{pmatrix} \begin{pmatrix} x \\ \tilde{x} \end{pmatrix} + \begin{pmatrix} B \\ 0 \end{pmatrix} r$$
$$y = (C \quad 0) \begin{pmatrix} x \\ \tilde{x} \end{pmatrix}$$

The vector K is called the observer for the system.
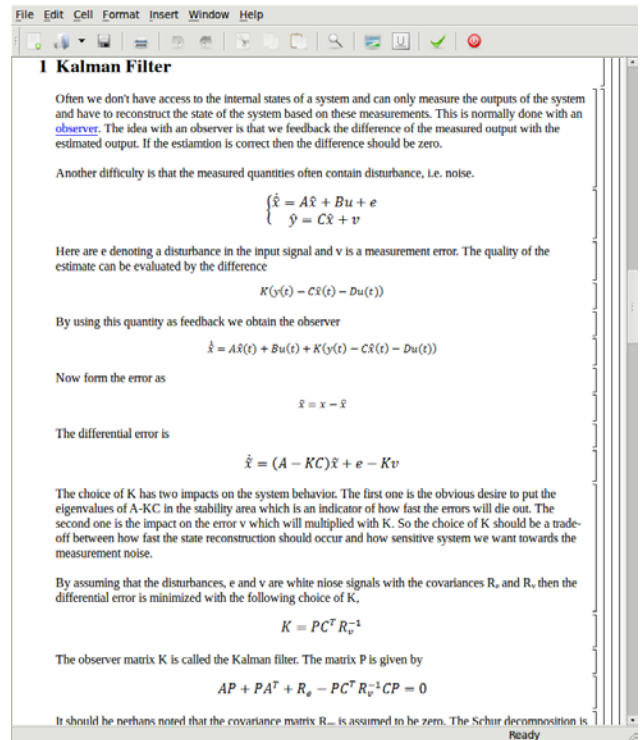

Figure 14. Kalman observer.

In real life systems the observed signals often contain noise. By introducing noise in the observed output signal the modeled system can be made more realistic. The `random` function is listed below:

```
type Seed = Real[3];

function random
  input Seed si;
  input Real tim;
  output Real x;
  output Seed so;
```

```
algorithm
  so[1] := abs(rem((171*si[1]*exp(
      mod(tim-11,tim+13)))),30269));
  so[2] := abs(rem((172*si[2]*exp(
      mod(tim-5,tim+7)))),30307));
  so[3] := abs(rem((170*si[3]*exp(
      mod(tim-23,tim+76)))),30323));
  if so[1] < 1e-4 then
     so[1] := 1;
  end if;
  if so[2] < 1e-4 then
     so[2] := 1;
  end if;
  if so[3] < 1e-4 then
     so[3] := 1;
  end if;
  x := rem((so[1]/30269.0 +so[2]/30307.0 +
     so[3]/30323.0),1.0);
end random;
```

The `time` input is needed to ensure that the Modelica compilers shouldn't consider the above function as constant.

Now the state-space model from the Modelica standard library can be re-written containing noise:

```
block StateSpaceWithNoise "Linear state
      space system with noise"
  parameter Real A[:,size(A, 1)] =
    {{0,1},{1,0}} ;
  parameter Real B[size(A, 1),:] =
    {{0},{1}};
  parameter Real C[:,size(A, 1)] =
    {{1,0}};
  parameter Real F[size(A, 1),:] =
    {{1},{0}};
  parameter Real D[size(C, 1),size(B, 2)]
    = zeros(size(C, 1), size(B, 2));
  extends Modelica.Blocks.Interfaces.MIMO(
     final nin = size(B, 2), final nout =
     size(C, 1));
  output Real x[size(A,1)] "State vector";
  Real si(start ={1,2,3});
  Real si2(start={11,27,127});
  Real randomE "input noise";
  Real randomV "measurement noise";
algorithm
  (randomE,si) := random(si,time/10);
  (randomV,si2) := random(si2,time/10);
equation
  der(x) = A * x + B * u + F*{randomE};
  y = C * x + D * u + {randomV};
end StateSpaceWithNoise;
```

```
model StateSpaceNoise
  StateSpaceWithNoise stateSpace;
  Modelica.Blocks.Sources.Exponentials
    ref(outMax=4,riseTime=1,
    riseTimeConst=1,fallTimeConst=0.2,
    offset=0,startTime=-1);
initial equation
  stateSpace.x[1]=1;
equation
  connect(ref.y, stateSpace.u[1]);
end StateSpaceNoise;
```

Lets now look at a simple noisy pendulum model where the output angle is observed with a Kalman observer:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix}\mathrm{u}$$
$$y = x_1$$

```
model KalmanFeedback
  parameter Real A[:,size(A, 1)] =
    {{0,1},{1,0}} ;
  parameter Real B[size(A, 1),:] =
    {{0},{1}};
  parameter Real C[:,size(A, 1)] =
    {{1,0}};
  parameter Real[2,1] K = [2.4;3.4];
  parameter Real[1,2] L = [2.4,3.4];
  parameter Real[:,:] ABL = A-B*L;
  parameter Real[:,:] BL = B*L;
  parameter Real[:,:] Z =
    zeros(size(ABL,2),size(AKC,1));
  parameter Real[:,:] AKC = A-K*C;
  parameter Real[:,:] Anew = [0,1,0,0 ; -
    1.4, -3.4, 2.4,3.4; 0,0,-2.4,1;0,0,
    2.4,0];
  parameter Real[:,:] Bnew = [0;1;0;0];
  parameter Real[:,:] Fnew = [1;0;0;0];
  StateSpaceNoise Kalman(
    StateSpace.A=Anew,
    StateSpace.B=Bnew,
    StateSpace.C=[1,0,0,0],
    StateSpace.F = Fnew);
  StateSpaceNoise noKalman;
end KalmanFeedback;
```
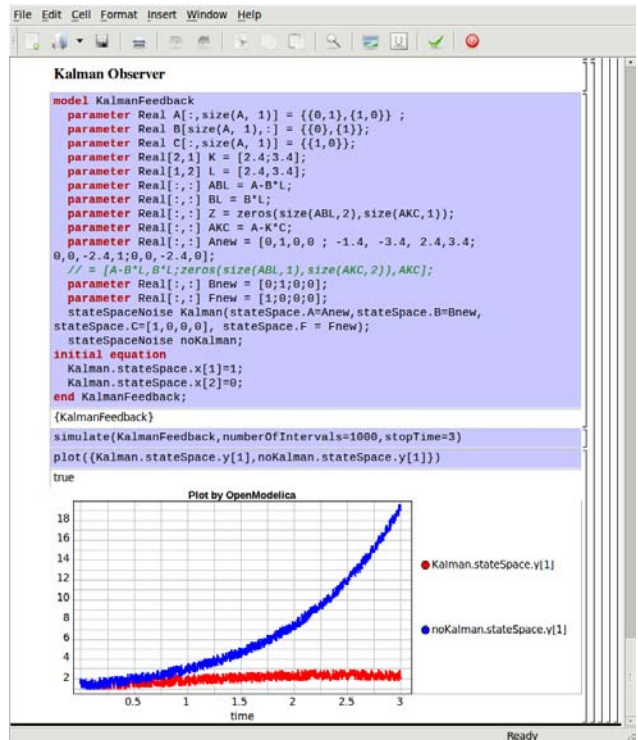

Figure 15. Pendulum angle control with Kalman observer.

3.6 Linear Quadratic Optimization

A good measure of suitable feedbacks, e.g. extra poles, is minimum of the input and output energy levels. Solving the minimum energy level functional leads to the algebraic Riccati equation, shown in Figure 16and Figure 17.
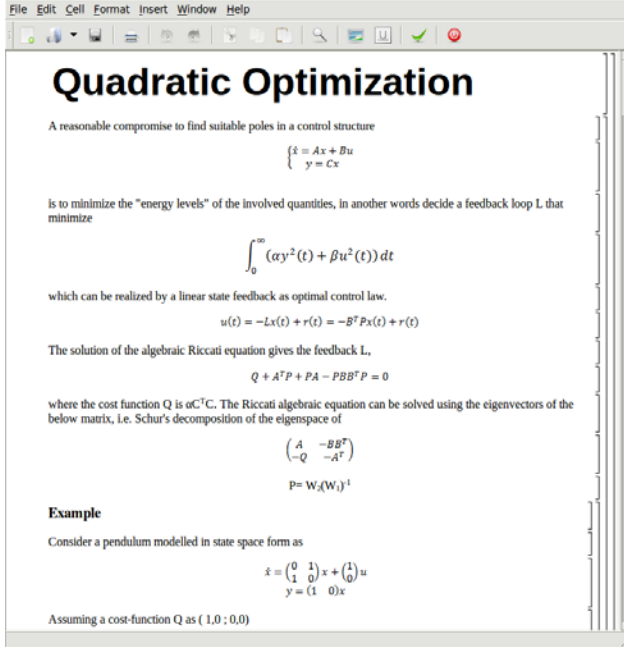

Figure 16. Quadratic optimization.

The algebraic Riccati equation is:

$$AP + PA^T + R_e - PC^T R_v^{-1} CP = 0$$

```
model RiccatiEq
  parameter Real A[2,2]=[0,1; 1,0];
  parameter Real B[2,1]=[0; 1];
  parameter Real C[1,2]=[1,0];
  Real P[2,2](start = Pinit);
  parameter Real Pinit[2,2] =
    [1,1.5;1.5,1];
  parameter Real Q[2,2] = [1, 0; 0, 0];
  Real L[1,2];
  Real L1 = L[1,1];
  Real L2 = L[1,2];
equation
  Q + P*A + transpose(A)*P -
     P*B*transpose(B)*P = [0,0;0,0];
  L = transpose(B)*P;
end RiccatiEq;
```
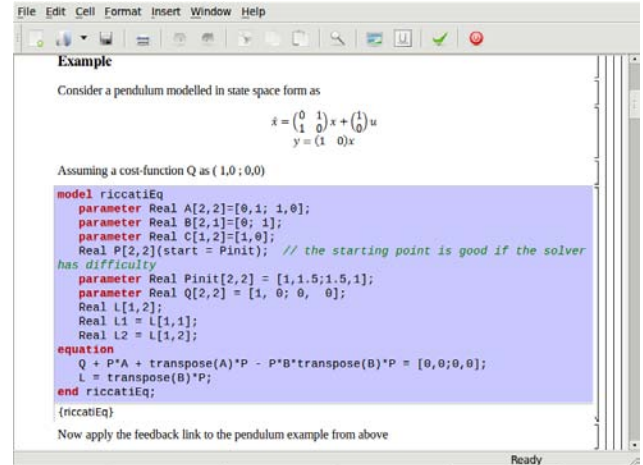

Figure 17. Solving Riccati equation with OpenModelica.

3.7 Linearization

Many nonlinear problems can be handled more easily by linearization around an equilibrium point. Lapapunov showed that if the linear approximation is stable then the nonlinear problem is also stable, at least around the equilibrium region. Thus we can investigate the behavior of the nonlinear system by analyzing the linearized approximation.

In the OpenModelica Compiler (OMC) a flag is introduced for linearization:

```
setCommandLineOptions({"+d=linearization"})
```

Assume that we have a non-linear two tank model shown below:

```
model TwoFlatTankModel
  Real h1(start = 2);
  Real h2(start = 1);
  Real F1;
  parameter Real A1 = 2,A2 = 0.5;
  parameter Real R1 = 2,R2 = 1;
  input Real F;
  output Real F2;
equation
  der(h1) = (F/A1) - (F1/A1);
  der(h2) = (F1/A2) - (F2/A2);
  F1 = R1 * sqrt(h1-h2);
  F2 = R2 * sqrt(h2);
end TwoFlatTankModel;
```

The output C-files are now generated with the linearization flag. By running the executable with the time argument the linearized model is generated which can be simulated:

```
buildModel(TwoFlatTankModel) //OMC
system("TwoFlatTankModel.exe -l 0.0 -v
   >log.out")
readFile("log.out")
```

The file `log.out` contains now the linearized model:

```
model Linear_TwoFlatTankModel
  parameter Integer n = 2; // states
  parameter Integer k = 1;
  parameter Integer l = 1;
  parameter Real x0[2] = {2,1};
  parameter Real u0[1] = {0};
  parameter Real A[2,2] = [-0.5,0.5;2,-3];
  parameter Real B[2,1] = [0.5;0];
  parameter Real C[1,2] = [0,0.5];
  parameter Real D[1,1] = [0];
  Real x[2](start = x0);
  input Real u[1](start = u0);
  output Real y[1];
  Real x_Ph1 = x[1];
  Real x_Ph2 = x[2];
  Real u_PF = u[1];
  Real y_PF2 = y[1];
equation
  der(x) = A * x + B * u;
  y = C * x + D * u;
end Linear_TwoFlatTankModel;
```
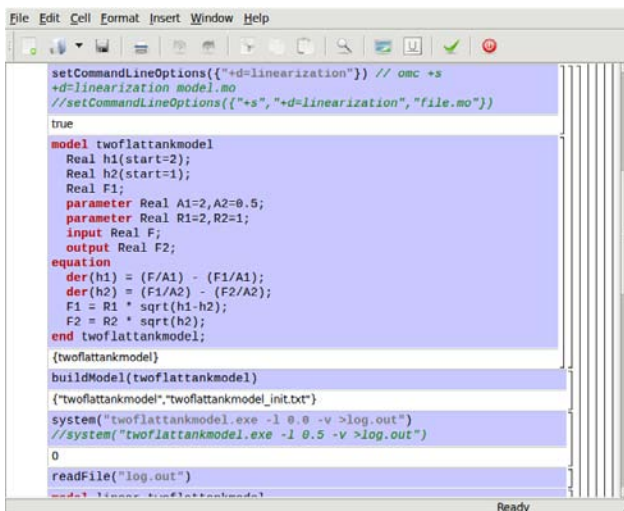

Figure 18. Linearization.

4 Other OMNotebook Applications

OMNotebook is also used for teaching modeling with Modelica (DrModelica), and programming in Scheme (DrScheme).

4.1 DrModelica

The existence of numerical algorithms and solvers are important aspects of equation-based environments such as Modelica tools.

OMNotebook is currently being used for course material (DrModelica) in teaching the Modelica language and equation-based object-oriented modeling and simulation, (see Figure 19).

It can easily be adapted to electronic books teaching other programming languages, such as Scheme (Sec-

tion 4.2). OMNotebook can also easily be used in other areas such as physics, biology chemistry, biomechanics etc., where phenomena can be illustrated by dynamic simulation within the book.
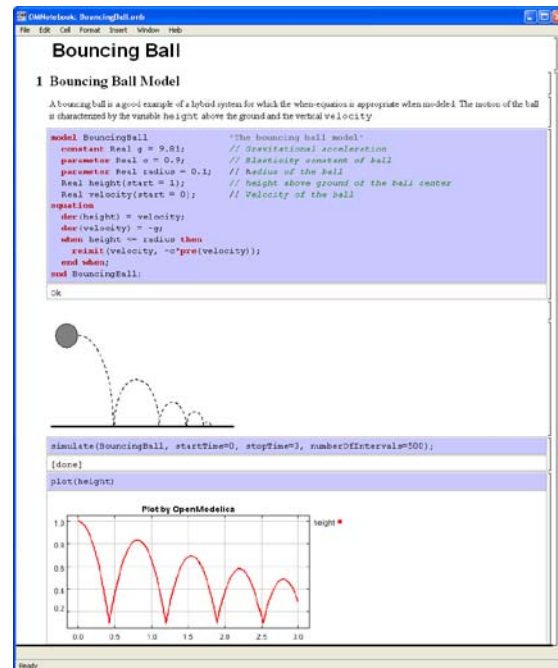

Figure 19. Bouncing ball example with movement animation in OMNotebook.

4.2 OMScheme

With OMScheme the OMNotebook paradigm is generalized towards other programming languages than Modelica, e.g. the Scheme programming language, [6]. An implementation of the factorial function using OMScheme is shown in Figure 20.
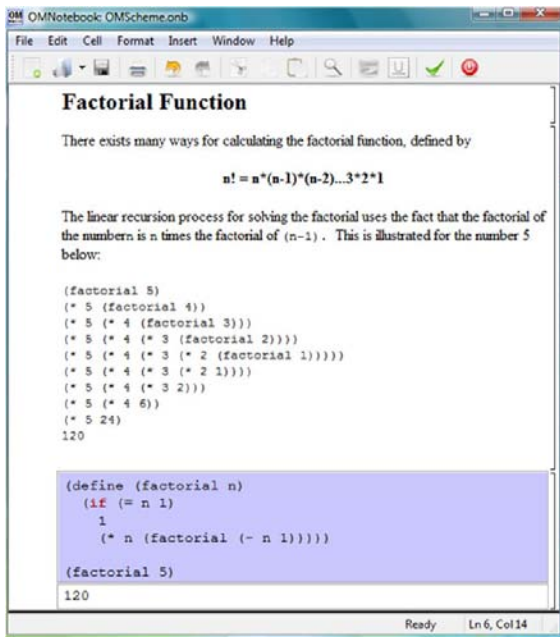
Figure 20. Factorial function in OMScheme.

5 Future Work

The inherent features of the Modelica language makes the next mile-stone choice quite natural, namely the adaption of the presented concept into other engineering courses as well, e.g. a future course material called DrMechanics for teaching the basics of mechanical systems.

A future generation of OMNotebook is planned to be extended to become available through a web applet which would make the material available without needing installation of any software.

One thing that is intentionally left out in this paper is frequency domain analysis, e.g. bode diagram. This is partly due to the inherent properties of the Modelica language, which is quite time domain dominant in its modeling style. A work-around was shown in this paper when studying the weight function and step response in time domain.

6 Conclusions

The OMNotebook is one of the first open source efforts offering interactive electronic books for teaching and learning modeling and programming.

In this paper we present its use in an active electronic book called DrControl for teaching control theory and applications.

The idea of active electronic books in OpenModelica has so far been employed in the two E-courses DrModelica and DrControl used successfully in graduate and workshop courses.

7 Acknowledgements

References

[1] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*, 940 pages, Wiley-IEEE Press, 2004.

[2] Modelica Association. *The Modelica Language Specification Version 3.2*, May 2010. http://www.modelica.org.

[3] Anders Sandholm, Peter Fritzson, Varun Arora, Scott Delp, Göran Petersson, and Jessica Rose. The Gait E-Book - Development of Effective Participatory Learning using Simulation and Active Electronic Books. In *Proceedings of the 11th Mediterranean Conference on Medical and Biological Engineering and Computing (Medicon'2007)*, Ljubljana, Slovenia, June 26 - 30, 2007.

[4] Eva-Lena Lengquist Sandelin, Susanna Monemar, Peter Fritzson, Peter Bunus. DrModelica – A Web-based Teaching Environment for Modelica, In *Proc. of the 44th Scandinavian Conference on Simulation and Modeling (SIMS2003)*, Västerås, Sweden, 2003

[5] Anders Fernström, Ingemar Axelsson, Peter Fritzson, Anders Sandholm, Adrian Pop. OMNotebook – Interactive WYSIWYG Book Software for Teaching Programming. In *Proc. of the Workshop on Developing Computer Science Education – How Can It Be Done*?. Linköping University, Dept. Computer & Inf. Science, Linköping, Sweden, March 10, 2006

[6] Mohsen Torabzadeh-Tari, Peter Fritzson, Adrian Pop, Martin Sjölund, *Generalization of an Active Electronic Notebook for Teaching Multiple Programming Languages* IEEE EDUCON Education Engineering 2010 – The Future of Global Learning Engineering Education, Madrid, Spain, 2010

[7] http://www.openmodelica.org [accessed 2011-02-03]