Modelica as a design tool for hardware-in-the-loop simulation

Marco Bonvini[a],Filippo Donida[b], Alberto Leva
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio 34/5, 20133 Milano, Italy
{leva,donida}@elet.polimi.it
[a]Graduate student at the Dipartimento di Elettronica e Informazione
[b]PhD student at the Dipartimento di Elettronica e Informazione

August 21, 2009

Abstract

This paper focuses on the automatic generation of microcontroller code for hardware-in-the-loop simulation using Modelica models. In this work a test is presented and commented in which Modelica is used to specify a control system, the inline integration code is obtained from the Modelica model and executed part on a PC, and part on a microcontroller board. The presented application, albeit created basically for educational purposes, covers quite different *scenarii*, therefore evidencing the usefulness of Modelica in the addressed context, and providing as a consequence some future research directions.

The contribution aspect of this work is twofold: on one side the *entire* cross-compilator software chain is built within the same framework; on the other hand, all the involved software tools are open-source (mainly GPL) licensed, making the application extremely modular and extensible. Furthermore this work will be included in the next release of the open source Modelica environment SimForge [3], thus enriching its Modelica back-ends support.

1 Introduction

This paper presents some experimental results relative to the usage of Modelica as modelling and specification language for hardware-in-the-loop simulation aims at control design.

The topic is of high interest both from the methodological and the application-oriented points of view. For the latter, it is evident that the usage of a single tool for the simulation of a control system irrespective of the code being run on the same computer as the model or on the final control architecture is of great help for the designer. Similarly, also having the process simulator running on dedicated hardware can help, particularly if the control strategy is the main object of the simulation studies. Also, having the possibility of deploying process simulation code on dedicated hardware could be of interest for testing control strategies directly on the field, where a PC may not be available.

From the methodological point of view, the envisaged activity apparently requires to obtain optimised simulation code with inline integration, starting from potentially complex object oriented models, in an user-transparent and reliable manner.

The importance having an integrated environment when testing the control performance in a closed loop simulated environment using an integrated environment was already well known in the 1996 [6], when the Control Aided Control System Design tools were used to off-line control design and HIL simulate the ABS systems. More in deep, in the automotive field the HIL Simulation is a quite established technique, both for engine control system test and design [8] and suspension control [2, 7]. Other works focus on the hydraulic servo position [9], machine tools and manufacturing systems [13] and spacecraft [11] control problem respectively. A quite detailed review on the benefits the HIL simulation can introduce for the design process is given in [5]. Moreover, as in [12], where a magnetic levitation device control is presented, the HIL simulation technique can also be used for educational scopes. In accordance with that work, this paper has strong educational intent and focuses on the possibility to use a complete (both software and hardware) GPL environment to control designing through the HIL simulation. The goal is to simulate a closed loop SCARA process. The model of the robot is obtained through the Lagrangian equations and formalized as a system of Differential Algebraic Equations, and the position controller is realised either as a continuous time system and as a digital algorithm[1]. The architecture is composed of a PC and a microcontroller board, communicating through the USB port. Different experiments — not reported here for space reasons — have been conducted where the two CPUs alternatively play the role of the process and of the control, the python language (and more precisely the python-visual library) being used on the PC as a 3d visualisation system.

The first part of the paper, section 2, briefly presents the Modelica model of the complete controlled system.

Section 3 describes the employed microcontroller board (namely the *Arduino*, [1] and its development systems, in connection with the Modelica environment. Convenient references are also provided to the interested reader in full detail.

Then, in section 4 it discussed how the various components of that system can be automatically turned into inline integration algorithms, so as to cover all the possible combination of microcontroller and PC in the simulation of the overall system.

Subsequently, in section 5, the configurations of the performed test is described, evidencing for each configuration which particular aspect of the research claims stated in the introduction is being investigated: for example, a test in which the microcontroller runs the simulator and the PC the control system is useful to analyse the latter together with the communications, while a test where the PC simulates the robot and the micro the control helps estimating the feasibility of a given algorithm on the target architecture, and so on. The test results are finally presented, ending with some configuration on the subsequent research activity.

[1]Only the discrete time version of the controller is treated in this paper.

2 The SCARA: model and control in Modelica

The SCARA robot we considered in this work in a two rotational degrees of freedom mechanical planar chain, actuated via two ideal electrical servos. The model of the robot is now introduced, starting from some considerations about the links masses.

Making the hypothesis of the mass of each link concentrated in a single point, the links inertias could be expressed as:

$$I_i = \frac{1}{12} m_i {a_i}^2 \tag{1}$$

with $_i = 1,2$ being the subscript for identifying the link, m the link mass, a the position of the center of mass with respect to the beginning of the link measured along the link itself.

From the Lagrangian equations, with some algebra, it is trivial to express the system of differential equations of the robot in the form $\tau = f\left(\theta, \dot{\theta}, \ddot{\theta}\right)$:

$$\tau = B(\theta)\,\theta + C\left(\theta, \dot{\theta}\right)\dot{\theta} + g(\theta) \tag{2}$$

where the inertia matrix $B(\theta)$ is:

$$B(\theta) = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix} \tag{3}$$

with the $B_{i,j}$ being:

$$\begin{aligned} B_{1,1} &= m_1 {l_1}^2 + I_1 + m_2 {a_1}^2 + m_2 {l_2}^2 + \\ &\quad + 2 m_2 a_1 l_2 \cos(\theta_2) + I_2 \\ B_{1,2} &= m_2 {l_2}^2 + m_2 a_1 L_2 \cos(\theta_2) + I_2 \\ B_{2,1} &= m_2 {l_2}^2 + m_2 a_1 l_2 \cos(\theta_2) + I_2 \\ B_{2,2} &= m_2 {l_2}^2 + I_2 \end{aligned} \tag{4}$$

the matrix corresponding to the centripetal, Coriolis, and viscous friction forces, $C\left(\theta, \dot{\theta}\right)$, is:

$$C\left(\theta, \dot{\theta}\right) = \begin{bmatrix} C_{1,1} & C_{1,2} C_{2,1} & C_{2,2} \end{bmatrix} \tag{5}$$

being the $C_{i,j}$ elements:

$$C_{1,1} = -2m_2a_1l_2\sin(\theta_2)\dot{\theta_2}$$
$$C_{1,2} = -m_2a_1l_2\sin(\dot{\theta_2})\dot{\theta_2}$$
$$C_{2,1} = m_2a_1l_2\sin(\theta_2)\dot{\theta_1}$$
$$C_{2,2} = 0 \qquad (6)$$

while the matrix for the effects of the gravitational filed, $g(\theta)$ is:

$$g(\theta) = 9.81\begin{bmatrix} g_{1,1} \\ g_{2,1} \end{bmatrix} \qquad (7)$$

with:

$$g_{1,1} = (m_1l_2 + m_2a_1)\cos(\theta_1) + m_2l_2\cos(\theta_1+\theta_2)$$
$$g_{2,1} = m_2l_2\cos(\theta_1+\theta_2) \qquad (8)$$

Considering also that the plane containing the robot joint space is orthogonal to the gravity direction, we can simplify the equation 2, thus obtaining:

$$\tau = B(\theta)\theta + C(\theta,\dot{\theta})\dot{\theta} \qquad (9)$$

Finally the "Lagrangian" Modelica model of the SCARA can be straightforward obtained from the previous DAE system implementing the equation 9 and specifying the torques (τ) as inputs while the angular positions (θ) as outputs.

A *more* object-oriented and `Modelica.Mechanics` based model for representing the robot had also been developed and mainly used for testing the correctness of the "Lagrangian" model. Unfortunately, at the time this experiment has been conducted, it has not been possible to use the OO Modelica model of the SCARA with the OpenModelica compiler, because of the current limitation in supporting the whole Modelica Standard Library.

The control model is a vectorial discrete-time Proportional Derivative regulator implemented as a Modelica algorithm with saturation on the control signal.

Moreover, to complete the control schema architecture, it has been necessary to implement a model for computing the robot inverse kinematic. The code is reported in the following.

```
model InverseKinematic
  import MBI = Modelica.Blocks.Interfaces;
  parameter Real L[2];//Lenght of links 1 and 2
  MBI.RealInput xy[2];//x & y positions
  MBI.RealOutput a[2];//joint space angles
algorithm
  a:=InverseKinematic(xy, L1, L2);
end InverseKinematic;

function InverseKinematic
  input Real xy[2];
  input Real L[2];
  output Real a[2];
protected
  Real x,y,c2,s2,c1,s1,a1,a2;
algorithm
   x  := xy[1];
   y  := xy[2];
   c2 := (x*x+y*y-L[1]*L[1]-L[2]*L[2])/(2*L[1]*L[2]);
   s2 := sqrt(1-c2*c2);
   a2 := atan2(s2,c2);
   c1 := ((L[1]+L[2]*c2)*x+L[2]*s2*y)/(x*x+y*y);
   s1 := ((L[1]+L[2]*c2)*y-L[2]*s2*x)/(x*x+y*y);
   a1 := atan2(s1,c1);
   a  := {a1,a2};
end InverseKinematic;
```

3 The Arduino microcontroller

Arduino is an open-source microcontroller electronic platform, featuring 14 digital pins that can be used both as input or output and 9 analog input pins. The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). In figure 1 the ArduinoDuemilanove microcontroller is shown.

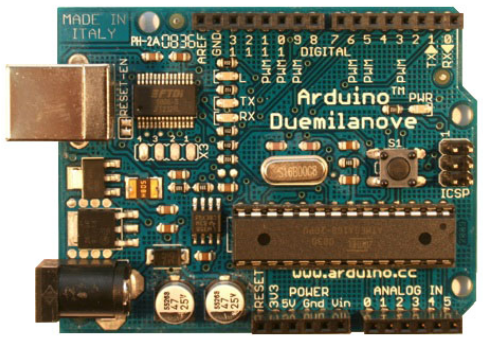

Figure 1: The Aurduino Duemilanove microcontroller.

Arduino executables can be a stand-alone processes or can communicate with other processes, i.e. running processes on a PC. Usually the structure of an Arduino program is quite standardized and is organized in four main parts:

- the *Preamble* section: collects all the includes and definitions statements,
- the *Variables and functions definitions* section. According to the data types of the Arduino language, all the variables used within the program must be declared here and, if necessary, instantiated. In addition the user-defined functions headers must be listed in this section.
- the *Setup* function setup(): this is called when your program starts, initializing the user-defined variables, pin modes, etc. The setup function runs only once, after each powerup or reset of the Arduino board.
- the *Loop* function loop(): this function contains the code to be sequentially executed, iteratively until resetting the Arduino.

With the intention to better explain the Arduino coding procedure, in the following a simple program for that board has been reported.

```
int buttonPin = 3;

// setup initializes serial and the button pin
void setup(){
  beginSerial(9600);
  pinMode(buttonPin, INPUT);
}

// loop checks the button pin each time,
// and will send serial if it is pressed
void loop(){
  if (digitalRead(buttonPin) == HIGH)
    serialWrite('H');
  else
    serialWrite('L');

  delay(1000);
}
```

As one could easily evince from the code lines reported above, the program is very simple, it just makes the Arduino sending a 'H' or 'L' character according to the state of a button (pressed or released).

4 Code generation

In this section a cross-compiler software to automatically translate the Modelica code into Arduino microcontroller code is presented. The code generation procedure, reported in figure 2 involves four different software layers: the OpenModelica compiler (in fucsia), the Java environment (light green), the Arduino environment (yellow) and the Maxima [10] symbolic manipulator (grey). In this section a description of the whole procedure is reported, for more details please refer to the SimForge documentation.

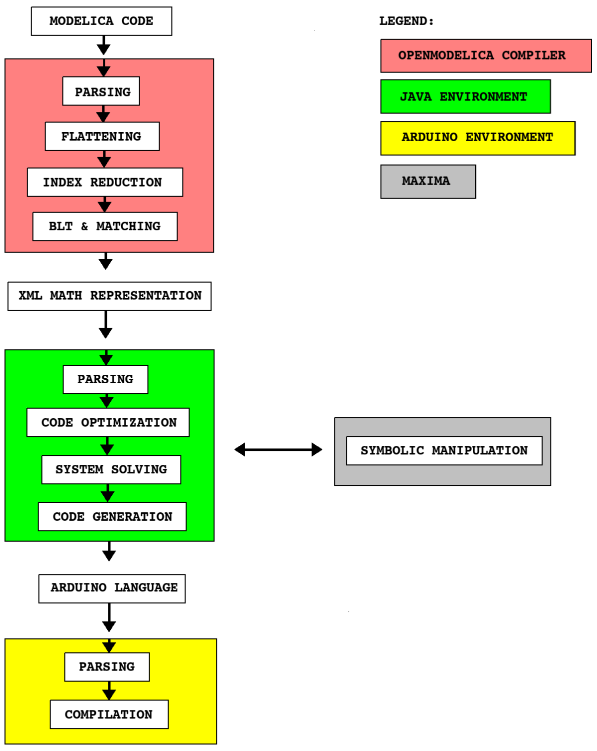

Figure 2: The steps for the Aurduino code generation from Modelica model.

The first step of the procedure is performed by the OpenModelica compiler that reads the Modelica file of the model, parses the Modelica source and creates the corresponding Abstract Syntax Tree (AST). After doing that, the tree is traversed for flattening the model, the index reduction algorithm is performed (if necessary) and the Tarjan algorithm is operated, thus obtaining the BLT structure and the variables-equations matching. At this stage the model is represented as an index-one system of differential algebraic equations and can be dumped out from the OpenModelica compiler through the XML dump module. This module of the compiler has been implemented with the intention to provide a standardized way for representing such a system of differential algebraic equations. To make the XML source syntactically constrained, an *ad hoc* XML Schema [4] has been created, requiring that it contains the variables lists (unknown, known and external), the equations lists (equations, algorithms, zero crossings, simplified equations, . . .) and, optionally, information for solving the system, i.e. the BLT structure and/or the matching algorithm output.

To make an example, if considering the following Modelica model:

```
model test_equation
  Real x(start = 1);
  parameter Real a = -1;
equation
  der(x) = a*x;
end test_equation;
```

a XML-equivalent representation could be[2]:

```
<?xml version="1.0" encoding="UTF-8"?>
<dae xmlns:p1="http://www.w3.org/1998/Math/MathML"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation=...
           ..."http://home.dei.polimi.it/donida...
           .../Projects/AutoEdit/Images/DAE.xsd">

<variables dimension="2">

 <orderedVariables dimension="1">
  <variablesList>
   <variable id="1" name="x" variability=...
   ..."continuousState" direction="none" ...
   ...type="Real" index="-1" origName="x" ...
   ...fixed="true" flow="NonConnector" ...
   ...stream="NonStreamConnector">
    <classesNames>
    <element> test_equation </element>
    </classesNames>
    <attributesValues>
      <initialValue string="1.0"> </initialValue>
    </attributesValues>
    </variable>
  </variablesList>
 </orderedVariables>

 <knownVariables dimension="1">
  <variablesList>
   <variable id="1" name="a" variability=...
   ..."parameter" direction="none" ...
   ...type="Real" index="-1" origName="a" ...
   ...fixed="true" flow="NonConnector" ...
   ...stream="NonStreamConnector">
    <bindValueExpression>
      <bindExpression string="-1"> </bindExpression>
    </bindValueExpression>
    <classesNames>
    <element> test_equation </element>
    </classesNames>
    </variable>
  </variablesList>
 </knownVariables>
</variables>

<equations dimension="1">
    <residualEquation id="1">der(x) - a * x = 0
    </residualEquation>
</equations>

<additionalInfo>
    <solvingInfo>
    <matchingAlgorithm>
      <solvedIn variableId="1" equationId="1" />
    </matchingAlgorithm>
    <bltRepresentation>
      <bltBlock id="1">
         <involvedEquation equationId="1" />
      </bltBlock>
    </bltRepresentation>
    </solvingInfo>
</additionalInfo>

</dae>
```

[2]Given a Modelica model, the XML representation is not unique, since some parameters can optionally be specified when dumping the model (i.e. if add the solving information, if dump the equations using the residual form, etc.), thus changing the content.

The XML representation reported in 4 is quite intuitive: at the top the header specifies where to find all the related schemes, then the `dae` tag contains all the variables, the equations and the additional information. The variables list is usually[3] split into two separated list: the ordered variables and the known variables lists respectively, the former containing all the state (also dummy states) and algebraic variables, while the latter listing all the parameters and constants. The second section is the equations section, and then, at the bottom of the file, the additional information are located, showing the matching algorithm output as well as the BLT representation of the system.

This representation is extremely useful since offers a standard machine-readable exchange format for the DAE system.

In next macro-step of the cross-compilation procedure, the XML file is processed through an *ad hoc* implemented Java routine. This software layer has two main goals:

- implement some xml functionalities to handle with the XML representation of the DAE system and
- provide some basic symbolic manipulation capabilities through the Java-Maxima interface.

[3]In some cases also the external variables and/or the external classes lists could be present. We invite the interested reader to refer to [4].

The Java module execution chain is basically made up of four sub-steps:

- First the DAE XML file is parsed and all the classes corresponding to the mathematical entities (variables, equations, matching output, BLT blocks, states selection,...) are instantiated within the Java environment,
- then the system of equations is re-formulated for the code generation, trying to use the matching algorithm output to solve the equations system. It can happen that the greater block of the BLT matrix is not a scalar. If this happen there are two possibilities: the Maxima interface can handle with the problem, thus solving the system for the variables specified from the matching algorithm output, otherwise (when Maxima can find more than a solution or none at all) the system must be included within the real code, together with a Newton procedure to iteratively found the solution, at each time step. As the reader can easily imagine, the latter scenario is not really desirable, since there is no *a priori* guarantee for the convergence of the Newton algorithm within the given time step.
- Thirdly the inlining procedure is applied to all the state equations, according to the specific integration algorithm (forward Euler, backward Euler, trapezium) and the obtained discrete-time system is solved with respect to the new discrete states variables.
- Finally the file required to compile the model of the controller within the Arduino environment is generated.

5 The SCARA: HIL

In this section the HIL experiment is described. As already introduced in section 1, the Arduino microcontroller runs the control algorithm, while the PC creates the position set points, simulates the robot model and visualizes the SCARA robot through the Python script. More precisely, an algorithm on the PC generates the position set points that are sent to the Arduino through the USB port. The Arduino microcontroller, translates the position set points into angle set points, using the inverse kinematic block and then, according to the PD gains, the torques to apply to each joint. The control signals are finally sent back to the PC, that simulates and visualizes the SCARA movement.

In this work, the objective has been to make the robot drawing a star. To do that, the position set points generation function samples 12 points on a couple of concentric circles (6 equidistant points over the inner and the other 6 equidistant points over the outer circle respectively), thus obtaining a star path. After doing that, the set point trajectory is thicken specifying the number of sub-samples between a point and the next on the star path.

In figure 3 the Scilab visualisation window is reported, showing how the controlled robot follows the target trajectory.

The experiment refers to a simulated SCARA having the first link with a length of 1 [m] and a weight of 2 [Kg], while the second 0.6 [m] and 1 [Kg]. The discretization technique has been used is the Forward Euler method, with a fixed step of 0.1 [s]. A time of 2[s] has been chosen for moving from a point to the next of the set point trajectory.

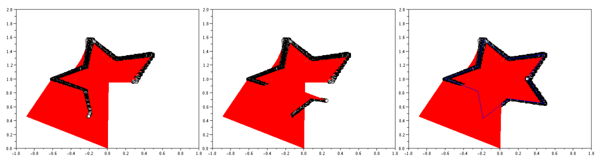

Figure 3: Graphical representation of the star drawing process of the closed loop system.

The Process Values (blue) and Set Points (red) of both the angles at top and Control Signals (green) at bottom for the star experiment are reported in figure 4 (the first angle is reported on the right column while the second angle on the left column).

Even if the two signals (PV and SP) seem to be quite overlapped, they are not identical, as it is possible to view from the figure 5, where a particular is shown.

In addition to that, a Python script has been implemented to 3d visualize the robot closed loop behavior. Figure 6 refers to a test case in which the simulation is done on the microcontroller, while the PC runs the control algorithm and the visualising machinery. In detail, the figure shows the animation window obtained with the python-visual

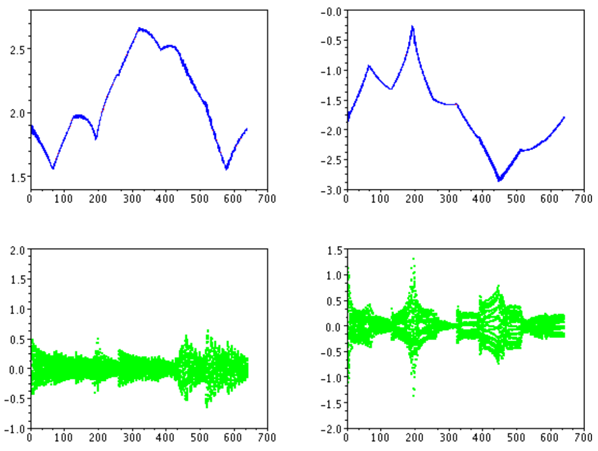
Figure 4: Process Values (blue) and Set Points (red) of both the angles at top and Control Signals (green) at bottom (the first angle on the right column while the second angle on the left column) for the star experiment.

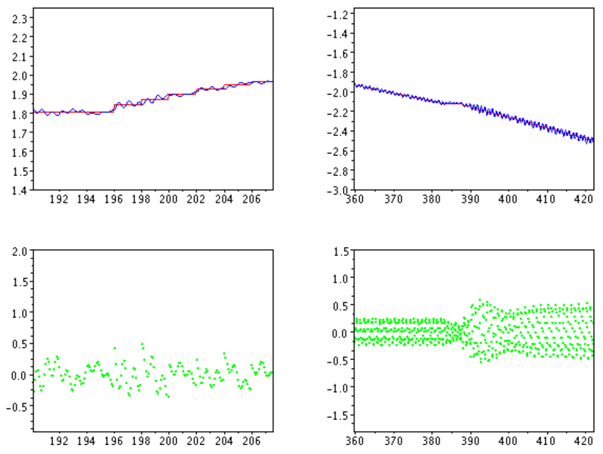
Figure 5: A particular of picture 4.

library for the online visualisation of the SCARA simulation.

All the Modelica, C (for the microcontroller), and python code will be made available as free software to the scientific community.

6 Conclusions

A closed-loop discrete-time SCARA process has been simulated with the HIL technique, with the Arduino microcontroller running the PD control, while the PC generating the trajectory, simulating the model of the robot and 3d visualising its movement.

Even if this work has mainly educational intentions, clearly shows the possibility of using an integrated GPL-licensed framework for automatically produce the HIL code from the Modelica language, for control aims. The openness of the presented framework is of great importance, specially for maintaining the modularity of the project, thus ensuring the scalability and the extensibility.

Future works will probably focus in two main directions: on one hand the possibility of specifying more complex control strategies will be inspected, on the other hand the eventuality of using a couple of Arduino microcontrollers for HIL simulation will be envisaged.

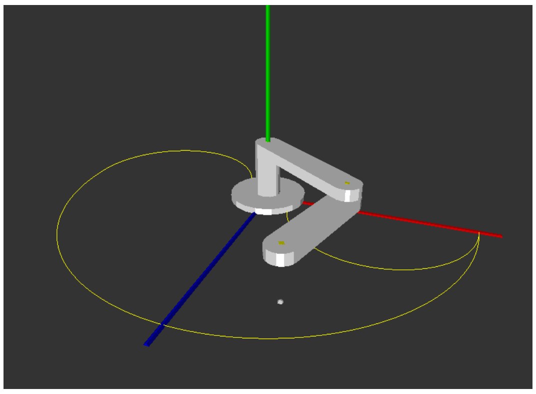
Figure 6: An example of python-based online animation of the simulation results obtained by the microcontroller, and fed to the visualising PC through the USB interface.

References

[1] The arduino home page. `http://www.arduino.cc`.

[2] S. B. Choi, Y. T. Choi, and D. W. Park. A Sliding Mode Control of a Full-Car Electrorheological Suspension System Via Hardware in-the-Loop Simulation. *Journal of Dnamic Systems, Measurements and Control*, 122:114–121, March 2000.

[3] Politecnico di Milano sede di Cremona. Simforge: a graphical modelica environ-

ment. `https://trac.wd.dei.polimi.it/simforge`.

[4] Filippo Donida. Xml schema for dae representation with the openmodelica compiler, 09 2009. `http://home.dei.polimi.it/donida/Projects/AutoEdit/Images/DAE.xsd`.

[5] R. Ernst. Codesign of embedded systems: status and trends. *Design & Test of Computers, IEEE*, 15(2):45–54, Apr-Jun 1998.

[6] H. Hanselmann. Hardware-in-the-Loop Simulation Testing and its Integration into a CACSD Toolset. In *IEEE International Symposium on Computer Aided Control System Design*, pages 152–156. IEEE, September 1996.

[7] K. S. Hong, H. C. Sohn, and J. K. hedrick. Modified Skyhook Control of Semi-Active Suspensions: A New Model, Gain Scheduling, and Hardware-in-the-loop Tuning. *Journal of Dnamic Systems, Measurements and Control*, 124:158–167, March 2002.

[8] R. Isermann, J. Schaffnit, and S. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, 7:643–653, August 1999.

[9] M. Linjama, T. Virvalo, J. Gustafsson, J. Lintula, V. Aaltonen, and M. Kivikoski. Hardware-in-the-loop environment for servo system controller design, tuning and testing. *Microprocessors and microsystems*, 24:13–21, December 2000.

[10] MIT. Maxima, a computer algebra system, 1960. `http://maxima.sourceforge.net/`.

[11] A. Ptak and K. Foundy. Real-time spacecraft simulation and hardware-in-the-loop testing. In *Real-Time Technology and Applications Symposium, 1998. Proceedings. Fourth IEEE*, pages 230–236, Jun 1998.

[12] P. S. Shiakolas and D. Piyabongkarn. Development of a Real-Time Digital Control System With a Hardware-in-the-Loop Magnetic Levitation Device for Reinforcement of Controls Education. *IEEE Transaction on Control Education*, 46(1):79–87, February 2003.

[13] G. Stoeppler, T. Menzel, and S. Douglas. Hardware-in-the-loop simulation of machine tools and manufacturing systems. *Computing & Control Engineering Journal*, 16(1):10–15, Feb.-March 2005.