PowerDEVS

Version 2.3

User's Guide

Ernesto Kofman and Federico Bergero

CIFASIS – CONICET
FCEIA, UNR, Argentina

2015

Contents

Chapter 1

Introduction to PowerDEVS

1.1 Features of PowerDEVS

PowerDEVS is an integrated tool modeling and simulation based on the Discrete Event System Specification (DEVS) formalism.

In spite of being a general purpose DEVS simulation tool, PowerDEVS is well suited for modeling and simulation of continuous and hybrid systems through the usage of Quantized State System (QSS) algorithms.

The main features of PowerDEVS are described below:

- Models can be built as hierarchical Block Diagrams using the *Model Editor* GUI.
- Atomic blocks can be taken from existing libraries and models or can be created using PowerDEVS *Atomic Editor* tool or any text editor.
- PowerDEVS libraries contain blocks for modeling continuous, hybrid and discrete time systems. For modeling classic discrete event systems, a Petri Net library is also provided.
- PowerDEVS can use Scilab as Workspace for defining model parameters, performing result analysis, etc.
- PowerDEVS can run under Linux–RTAI Operating System, allowing precise time synchronization, hardware interrupts access and different features suitable for PC based automatic control systems.
- PowerDEVS engine and models are programmed in C++.
- PowerDEVS is an Open Source multiplatform tool under General Public License (GPL).

- Source code as well as Windows and Linux binaries are available at SourceForge: `http://sourceforge.net/projects/powerdevs/`

1.2 PowerDEVS Architecture

PowerDEVS is composed by several independent programs:

- The *Model Editor*, which contains the main graphical user interface allowing the hierarchical block-diagram construction, library managing, parameter selection and other high level definitions as well as providing the linking with the other programs.
- The *Atomic Editor*, which permits editing PowerDEVS atomic models for elementary blocks by defining the corresponding DEVS transition, output and time advance functions using C++ language.
- The *Preprocessor*, which automatically translates the model editor files into C++ code to build up the simulation code. It also links the C++ code of the different atomic models and compiles it together with the C++ PowerDEVS engine to produce a stand alone executable file which simulates the system.
- The *Simulation Interface*, which runs the stand alone executable and permits to change different simulation parameters (experimental setup) such as final time, number of simulations to perform, and the simulation mode (normal simulation, timed simulation, step-by-step simulation, etc).
- An instance of Scilab, which acts as a workspace, where simulation parameters can be read, and results can be exported to.

1.3 Basic Usage

Simulating an existing model

Figure 1.1 shows the PowerDEVS Model Editor main window. A model can be opened from the `File` menu or using the `Ctrl+o` shortcut.

The model shown in Figure 1.1 corresponds to the control of a DC motor. It can be found at `powerdevs/examples/hybrid/dc_drive/dc_drive_buck.pdm`. PowerDEVS Model Editor saves the models with extension `.pdm`.

After opening the model, it can be simulated. Clicking on the blue *play* icon or going to `Simulation->Simulate` at the menu (or just pressing the

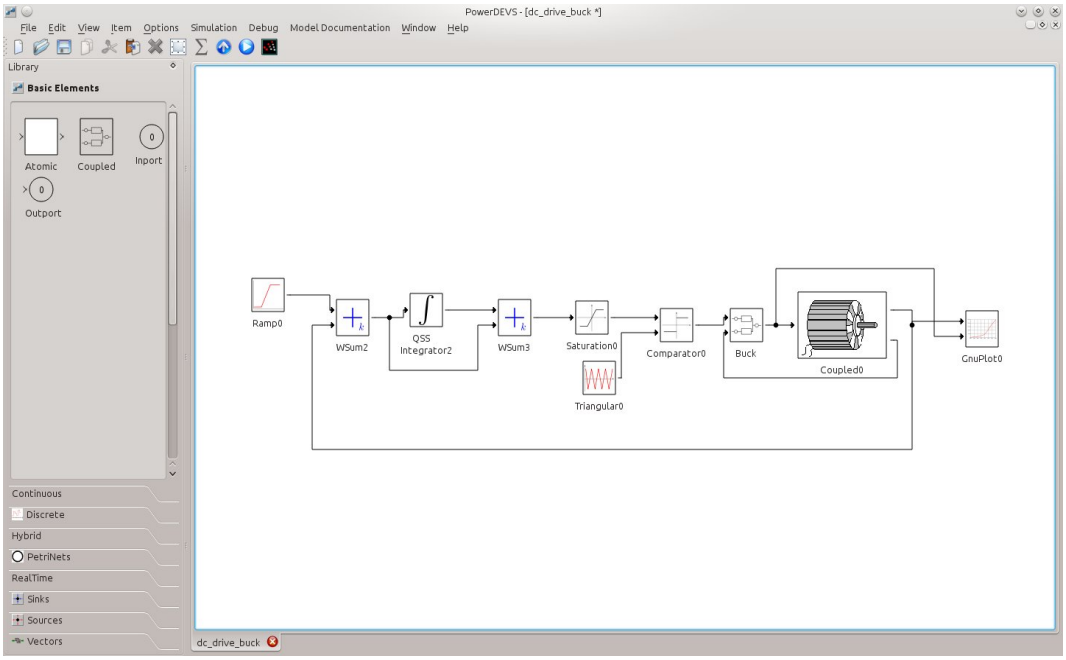

Figure 1.1: PowerDEVS Model Editor

shorcut key `F5`) PowerDEVS invokes the Preprocessor that automatically generates the C++ code and compiles it. Then, it opens the Simulation Interface, which can be seen at the left of Figure 1.2

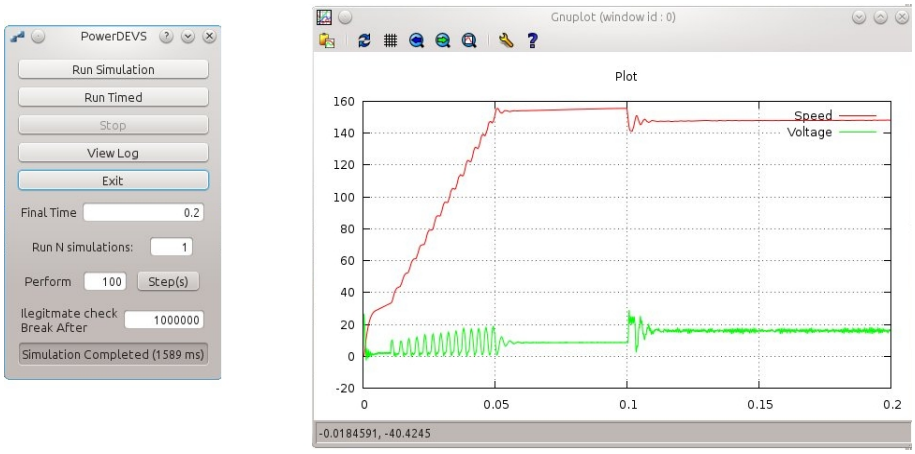

Figure 1.2: PowerDEVS Simulation Interface and Results

Clicking on the `Run Simulation` button the simulation is executed. The model in Figure 1.1 contains a `GNUPlot` block. This block opens an instance of Gnuplot to plot the signals it receives as it is shown at the right of Figure 1.2.

Creating a new model using existing blocks

Using the PowerDEVS Model Editor, a new model can be created from the `File` menu or using the `Ctrl+N` shortcut.

Suppose that you want to build a model that simulates the first order continuous time system

$$\dot{x}(t) = \sin(2 \cdot \pi \cdot t) - x(t)$$

A block diagram for the continuous time system consists in an *Integrator* (computing $x(t)$ from $\dot{x}(t)$), a *Sum* block that calculates $\sin(t) - x(t)$ and a source block that provides the input signal $\sin(t)$.

Figure 1.3 shows this block diagram in the PowerDEVS Model Editor. The blocks are drag and dropped from the libraries at the left. The `QSS_Integrator` and the `WSum` blocks were taken from the `Continuous` library. The `Sinusoidal` source block was taken from the `Sources` library and the `GNUPlot` block was taken from the `Sinks` library. Then, the blocks are connected clicking on an input port and dragging until reaching the corresponding output port. Connections can be also performed from output ports to input ports or from input ports to previously existing lines.

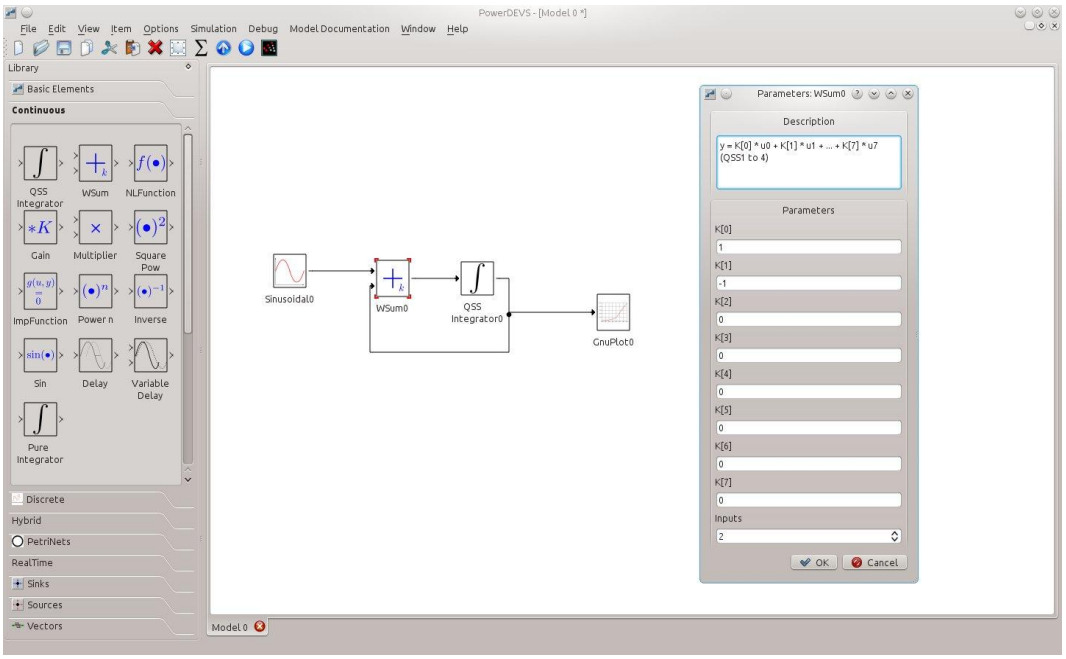

Figure 1.3: PowerDEVS Model Editor

Double–clicking on a block opens the corresponding parameter dialog window, where the block parameters can be modified. At the right of Figure 1.3

the parameters of the `Wsum` are shown. There, the second parameter (which corresponds to the gain of the second input port) was modified so it is -1 instead of 1.

After modifying the parameters, the model must be saved before it can be simulated.

1.4 Features of DEVS and QSS Algorithms

PowerDEVS represents and simulates its models based on the *Classic DEVS Formalism*. This formalism allows to represent any system performing a finite number of changes in a finite interval of time.

Thus, in theory, any discrete time or discrete event model can be represented by DEVS and so by PowerDEVS.

In the libraries of PowerDEVS we have included blocks to model arbitrary discrete time systems and also to model *Timed Petri Nets*. Yet, blocks for different discrete formalisms can be eventually included.

However, the main feature of PowerDEVS is that it has a complete library to simulate continuous and hybrid systems using *Quantized State System* (QSS) methods.

QSS methods perform numerical approximations of continuous time systems. These approximations can be expressed as DEVS models.

These algorithms, besides having some nice stability and error bound propeties, show noticeable advantages in the simulation of systems with frequent discontinuities.

In PowerDEVS, QSS methods are implemented in the `QSS_Integrator` block. This block receives and provokes events representing changes in piecewise polynomial trajectories. The remaining blocks of PowerDEVS continuous and hybrid libraries perform different math operations on these trajectories in order to build arbitrary block diagrams. Source and Sink blocks were also programmed in order to work with QSS methods.

Chapter 2

Basic Modeling with PowerDEVS

In this chapter, we explain how to build models using existing blocks from the libraries.

2.1 Building a Model

Let us suppose you want to build a model of a DC drive represented by the following equations:

$$\begin{aligned} \frac{di_a(t)}{dt} &= \frac{U_a(t) - R_a \cdot i_a(t) - K \cdot \omega(t)}{L_a} \\ \frac{d\omega(t)}{dt} &= \frac{K \cdot i_a(t) - b \cdot \omega(t) - \tau(t)}{J} \end{aligned} \tag{2.1}$$

where the state variables $i_a(t)$ and $\omega(t)$ are the armature current and speed, respectively. The parameters are R_a (armature resistance), L_a (inductance), b (friction), J (inertia), and K (electro-mechanical constant). The model has also two input variables: $U_a(t)$ (armature voltage) and $\tau(t)$ (load torque).

We shall use the following values for the parameters:

$$\begin{aligned} R_a = 1.73,\ L_a = 2.54 \times 10^{-3},\ J = 1.62 \times 10^{-5}, \\ b = 1.12 \times 10^{-5},\ K = 0.0551 \end{aligned} \tag{2.2}$$

Initially, we shall suppose that the input signals are steps:

$$U_a(t) = 24 \cdot \mu(t), \quad \tau(t) = 0.25 \cdot \mu(t - 0.1) \tag{2.3}$$

where $\mu(t)$ is the unit step (i.e., it takes the value 1 for all $t > 0$ and it is 0 otherwise).

The first step then consists in creating a new model (`Ctrl+N`) and building the block diagram using blocks from the libraries at the left of the modeling window.

We need blocks of type `QSS integrator` and `WSum` from the `Continuous` library. For the input signals, we take `Step` blocks from the `Source` library. In order to plot the results, we take a `GnuPlot` block from the `Sink` library.

Then, we can connect the blocks as shown in Fig.2.1.

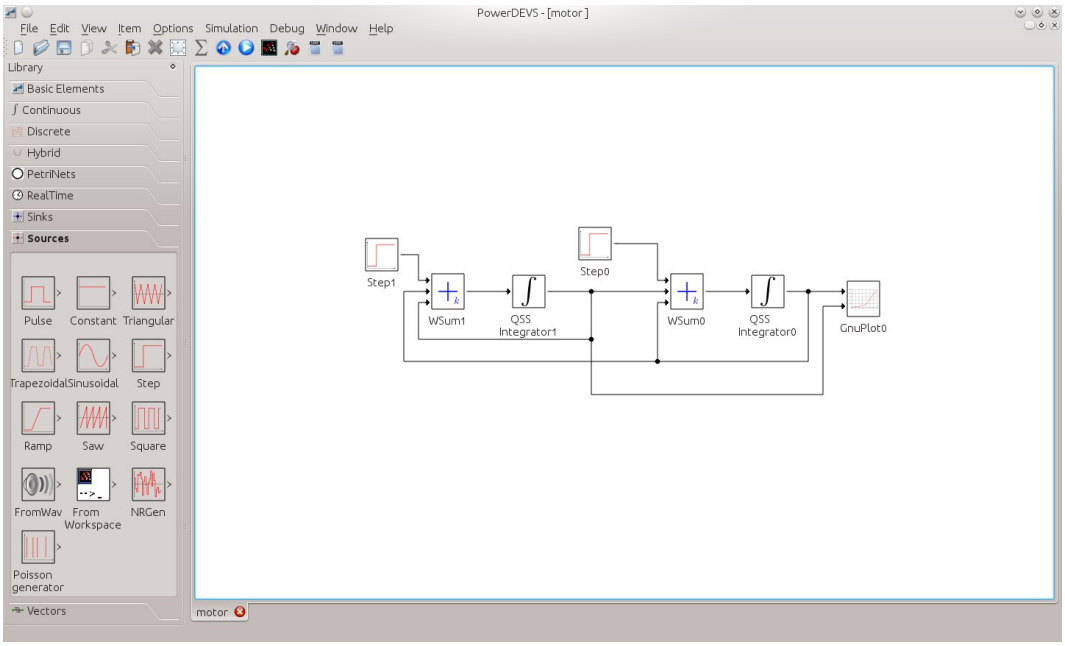

Figure 2.1: PowerDEVS Model of a DC Motor

Notice that the `WSum` block in the `Continuous` library has only two input ports, while the same block in the model has three input ports. In this block (and also in the `GnuPlot` block) the number of input ports is a parameter which can be changed by double clicking on the block.

Besides the number of input ports, the `WSum` and `Step` blocks have other parameters that must be changed in order to correctly represent the model of Eq.(2.1). Figure 2.2 shows the parametrization of these blocks.

As it can be noticed, most parameters in Figure 2.2 were defined by expressions like `K/J`. Most blocks of PowerDEVS libraries accept Scilab expressions as parameters.

If some variable involved in a parameter expression were not defined in Scilab, PowerDEVS will provoke a warning message during the simulation and the corresponding parameter will take the value `0`.

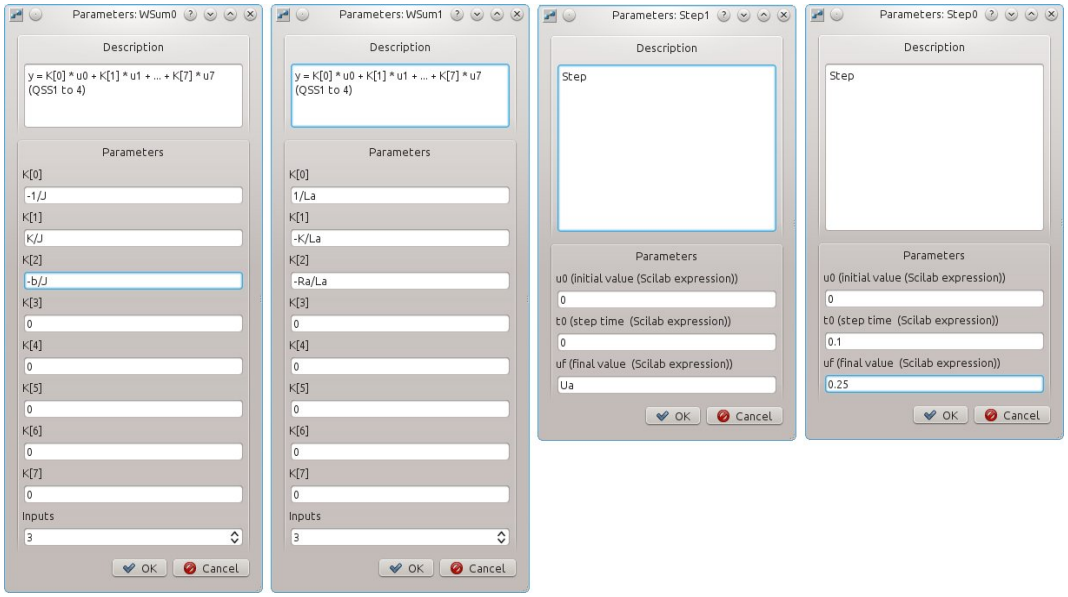

Figure 2.2: Block Parameters of the DC Motor Model

Scilab parameters can be also defined in PowerDEVS using the block `Scilab Command` from the `Sink` library. This block executes Scilab commands at the different stages of a simulation run. Thus, it can executes a command like `Ra=1.73` in order to set a value for variable `Ra`. Figure 2.3 shows the model with the parameters of this block.

In order to work properly, the block must be initialized before the other blocks, so when they ask Scilab for their parameters, the corresponding variables were already set. To that goal, PowerDEVS allows to establish priorities among the blocks, so that they are initialized in a desired order. Going to `Edit->Priority` (or clicking on the blue up arrow icon) a new window is open to select the model priorities. Figure 2.3 also shows the priority select window.

One problem with the model of Figure 2.1 is that the block names do not provide information about the variables they calculate. Block names can be changed by double–clicking on the corresponding label.

Additional information can be provided to the model by inserting *annotations*. Annotation labels can be added by selecting the option `Add annotation` after right–clicking on the model or through the `Edit` menu.

Figure 2.4 shows the DC Motor model with the annotations and the blocks renamed.

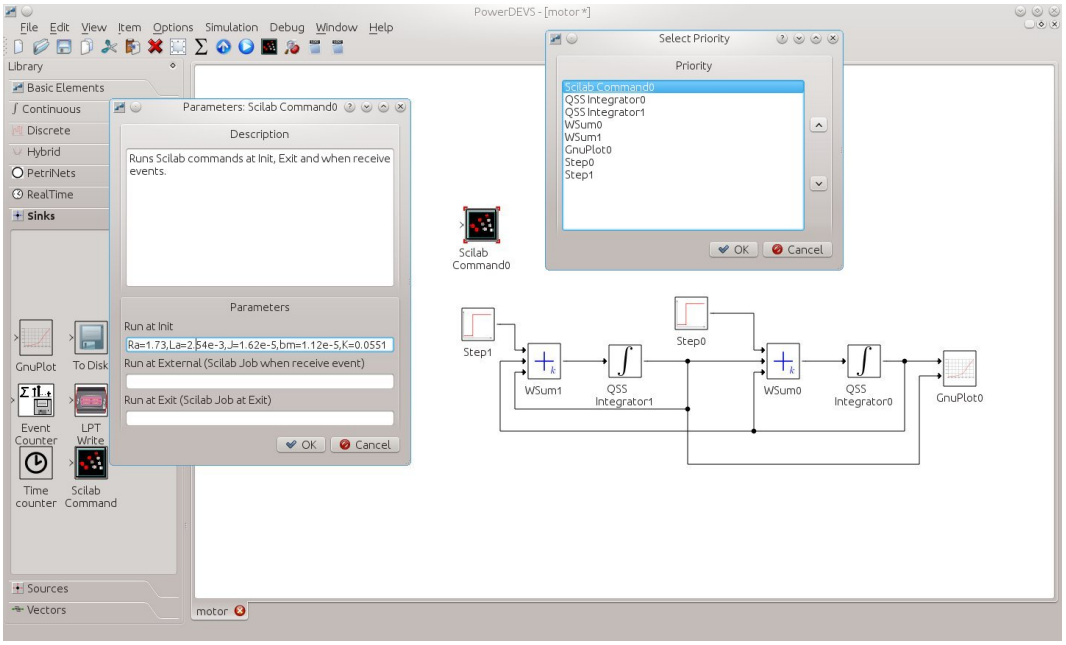

Figure 2.3: Scilab command block parameters and model priorities

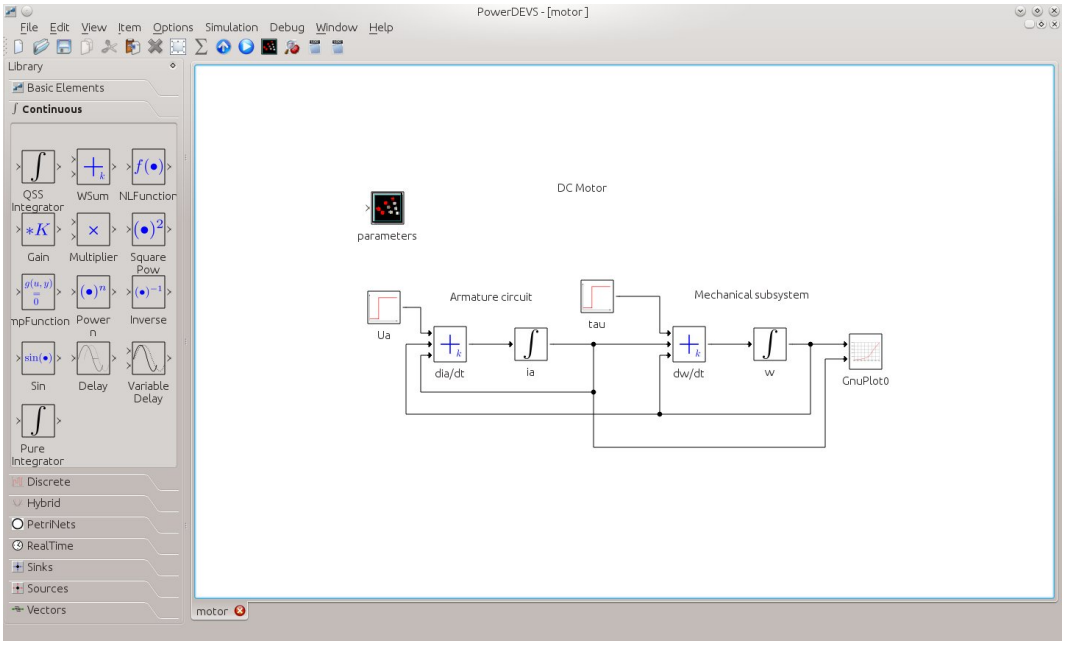

Figure 2.4: PowerDEVS Model of a DC Motor with annotations and renamed blocks

2.2 Hierarchical Coupling

Let us suppose that we want to implement a speed control for the motor model. For that goal, we propose the following *Proportional–Integral* (PI) control law:

$$\dot{e}_I = \omega_{\text{ref}}(t) - \omega(t)$$
$$U_a(t) = k_p \cdot \omega_{\text{ref}}(t) - \omega(t) + k_I \cdot e_I(t)$$

where $\omega_{\text{ref}}(t)$ is the reference speed trajectory.

This controller can be easily built using `Quantized Integrator` and `WSum` blocks, obtaining the model shown in Figure 2.5. There, the reference speed $\omega_{\text{ref}}(t)$is a ramp signal, generated by the `Ramp` block of the `Source` library.

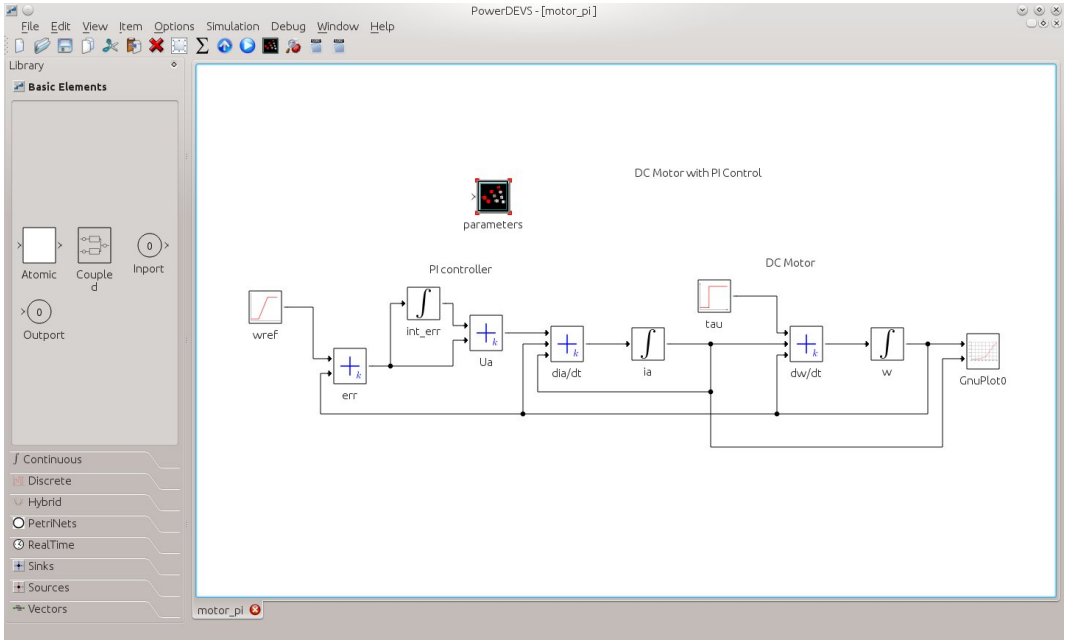

Figure 2.5: DC Motor with a Proportional Integral controller

This block diagram has a problem. It contain too many blocks and so it cannot be easily analyzed and understood.

In cases like this, the model organization can be improved by grouping the blocks of each sub–system to form *coupled* blocks that, at the higher hierarchical level, are seen as single blocks.

PowerDEVS allows to group blocks inside `Coupled` blocks at the `Basic Elements` library.

In this case, we shall group the whole DC motor model as a single coupled model. For that goal, we first drag a `Coupled` block to the model, and then, we shall select the blocks corresponding to the motor as shown in Fig.2.6.

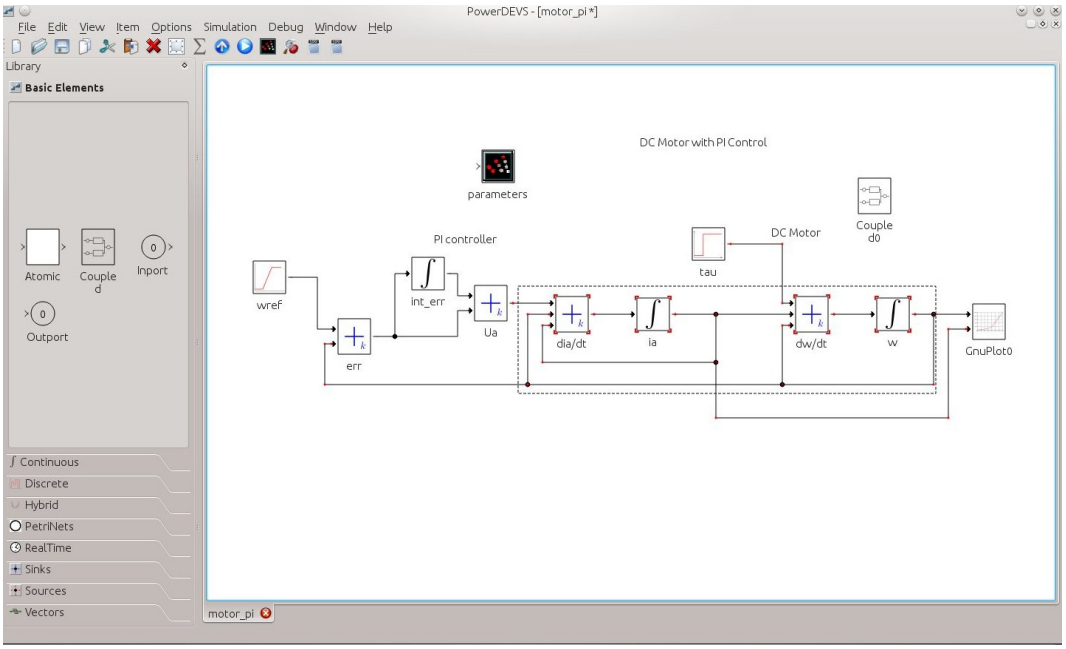

Figure 2.6: Creating a Coupled Model

Then, we shall cut the selected blocks with `Ctrl+x` or by clicking on the corresponding option at the `Edit` menu.

After that, we open the `Coupled` block by right clicking on it or from the `Edit` menu and then we paste the blocks that were cut from the whole model.

We also drag into this subsystem two `Inport` (input ports) and two `Outport` (output ports) blocks from the `Basic Elements` library to connect the corresponding variables. After renaming the blocks accordingly, the subsystem looks like the one shown in Fig.2.7.

Once the subsystem is complete, it can be used at the higher hierarchical layer as a single block. Figure 2.8 shows the resulting model, where the coupled block was renamed as `DC Motor`.

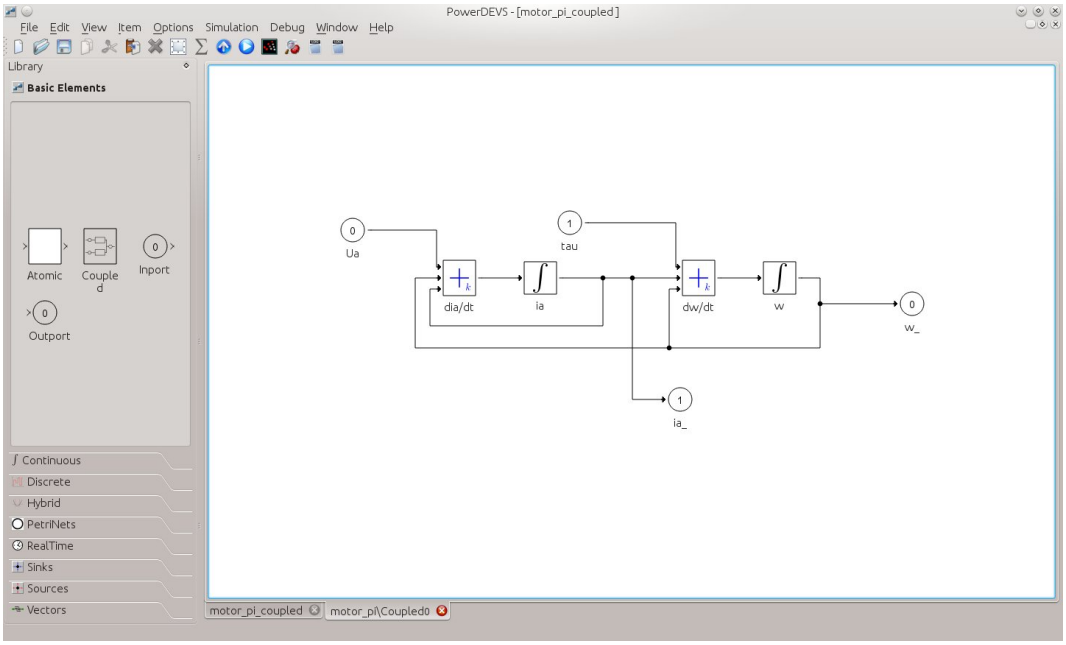

Figure 2.7: DC Motor Subsystem

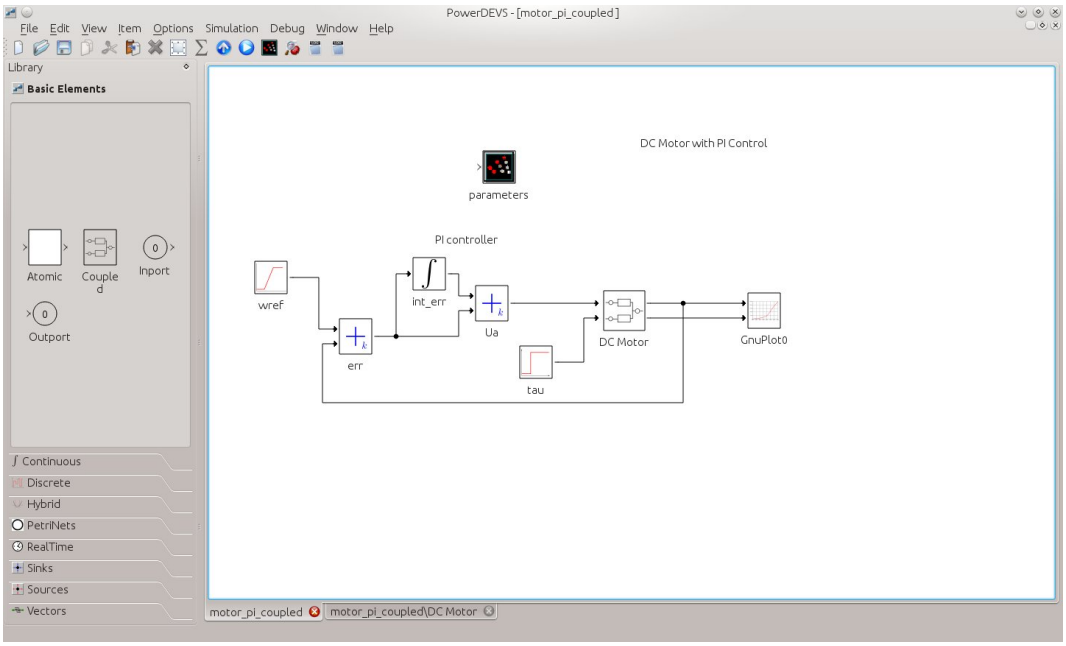

Figure 2.8: Controlled DC Motor using hierarchical coupling

Chapter 3

Simulation with PowerDEVS

In this chapter, we explain the different options offered by PowerDEVS to simulate a model.

3.1 The Simulation Procedure

The Preprocessor Module is in charge of translating the Block Diagram describing the model into C++ code. The result of this translation is a header file `model.h` and a makefile `Makefile.include` located both at the folder `powerdevs/build`.

Then, the Preprocessor invokes the C++ compiler which generates the executable file called `model` (or `model.exe` under Windows OS). The executable file is located at the folder `powerdevs/output`.

The simulation is then performed by invoking the `model` program with additional parameters.

The Preprocessor can be called directly from PowerDEVS main window at `Simulation->Simulate`, which generates the code, compiles it and opens the simulation interface.

3.2 Simulation from the Interface

The simulation interface (Fig.3.1) is a GUI for the `model` executable file generated by the Preprocessor.

It is invoked by clicking on the blue *play* icon, going to `Simulation->Simulate` at the menu, or just pressing the shorcut key `F5`.

Figure 3.1: PowerDEVS Simulation Interface

Before performing the simulation, the *final time* must be selected. Then, the simulation can be carried on in different ways:

- The button `Run Simulation` performs the simulation in the normal way.
- The button `Run Timed` simulates the system in a *Real Time* fashion, i.e., synchronizing the simulation with the physical clock.
- The button `Step(s)` performs N simulation steps, where N is the parameter entered at its left. This can be used for debugging purposes.

The simulation interface has also the following functions:

- The `Stop` button can be used to break a simulation execution.
- The `View Log` button opens the log file of the last simulation. This log file informs the CPU time taken by the simulation, as well as the debug information of the different blocks.
- The `Exit` button closes the simulation interface.
- The `Run N simulations` option permits selecting the number of times a simulation is executed. Multiple executions of a simulation can be useful in presence of random signals or parameters in order to obtain statistics.
- The `Illegitimate Check` option permits selecting the maximum allowed number of simulation steps without advancing the simulation time. If this condition is detected, PowerDEVS aborts the simulation as it considers that the model is *illegitimate*.

It is important to take into account that the simulation interface does not compile the model. It only runs the last compiled model.

Thus, if a model is modified, the changes will not be taken into account until the model is recompiled from PowerDEVS main window by invoking again the Preprocessor with the play button, with `Simulation->Simulate`, or with the shortcut `F5`.

However, if a model contains Scilab parameters, their changes at Scilab workspace are taken into account without the need of recompiling.

3.3 Simulation from the Command Line

The simplest way of executing a simulation is making use of the simulation interface, as explained above. However, in certain applications, it could be of interest to run the simulations from the command line.

For that goal, the preprocessor must be invoked (play button, `Simulation->Simulate`, or `F5`) in order to get the model compiled.

Then, the simulation can be performed by invoking the executable file `model` (or `model.exe` under Windows OS) from the command line with the following switches:

```
-i     Run intearctive shell
-tf t  Set final time to t (double format)
-ti t  Set initial time to t (double format)
-rt    Synchronize events with real time
-d     Debug simulation
-b n   Stop simulation after n steps in the same time (to detect illegitimate mode
```

The debug option `-d` prints the simulation engine activity performed at each step.

The interactive shell accessed through the `-i` switch allows to advance the simulation in a step by step fashion for debugging purposes. The shell commands are:

```
r: Run model
p: Show percentage done
y [0|1] sYnch events with real time
x n: Run n simulations
s [n]: Make n steps
o Stop simulation
t time: Set final Time
b n: Stop simulating after n steps in the same time (illegitimal)
q: quit
```

Notice that the options coincide with those of the GUI. However, when running from the command line, the debugging information can be directly seen at the console.

3.4 Simulation from Scilab

The Scilab command `simpd(tf)` simulates the last compiled model up to the final time `tf`.

Chapter 4

PoweDEVS and Scilab

Chapter 5

Realtime simulation

Appendix A

Engine functions

Appendix B

PowerDEVS libraries