INTRODUCTION TO OBJECT-ORIENTED MODELING AND SIMULATION WITH MODELICA USING THE OPENMODELICA ENVIRONMENT

Peter Fritzson[1], Adrian Pop[1], Peter Aronsson[2], Håkan Lundvall[1], David Broman[1], Daniel Hedberg[2], Jan Brugård[2]

[1]PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
[2]MathCore Engineering AB, Teknikringen 1F, SE-583 30, Linköping, Sweden

petfr@ida.liu.se (Peter Fritzson)

Abstract

Modelica is a modern, strongly typed, declarative, equation-based, and object-oriented language for modeling and simulation of complex systems. Major features are: ease of use, visual design of models with combination of lego-like predefined model building blocks, ability to define model libraries with reusable components, support for modeling and simulation of complex applications involving parts from several application domains, and many more useful facilities. This paper gives an overview of some aspects of the Modelica language and the OpenModelica environment – an open-source environment for modeling, simulation, and development of Modelica applications.

Keywords: Modelica, OpenModelica, Modeling, Equation-Based, Open Source

Main Author's biography:

Peter Fritzson is a Professor and Director of the Programming Environment Laboratory (PELAB), at the Department of Computer and Information Science, Linköping University, Sweden. Peter Fritzson is chairman of the Scandinavian Simulation Society, secretary of the European simulation organization, EuroSim; and vice chairman of the Modelica Association. His main area of interest is software engineering, esp. design, programming languages and environments, including modeling and simulation.

Presenting Authors' biographies:

Håkan Lundvall, MSc, is a PhD student at PELAB, Linköping University, Sweden. He is involved in the OpenModelica efforts since several years, and has among other things designed and implemented the hybrid and discrete event support. His main area of interest is generation of parallel code from Modelica, targeting clusters and multi-core architectures.

Jan Brugård, MSc, is the CEO of MathCore Engineering AB, Linköping, Sweden. He is an experienced modeler with several years of experience of modeling complex engineering applications, primarily mechanical and electrical applications, and is a driving force behind the development of the MathModelica System Designer product.

1 Introduction

A large number of modeling and simulation tools are available on the market. Most tools are very specialized and only address a specific application domain. These tools usually have incompatible model formats.

However, certain tools are of a more general nature. For modeling and simulation of dynamic systems, we have e.g. the following de-facto standards:

- Continuous: Matlab/Simulink,... [15], MatrixX/ SystemBuild, Scilab/Scicos, ACSL,... for general systems, SPICE and its derivates for electrical circuits, ADAMS, DADS/Motion, SimPack, SimMechanics,... for multi-body mechanical systems, ANSYS, FEMLAB, etc. for finite-element analysis, ...
- Discrete: general-purpose simulators based on the discrete-event GPSS line, VHDL- and Verilog simulators in digital electronics, etc.
- Hybrid (discrete + continuous): Modelica, gPROMS [2], AnyLogic, VHDL-AMS and Verilog-AMS simulators (not only for electronics but also for multi-physics problems), etc.

The insufficient power and generality of the former modeling languages has stimulated the development of Modelica (as a true object-oriented, multi-physics language) and VHDL-AMS/Verilog-AMS (multi-physics but primarily for electronics, and therefore often not general enough).

1.1 Modelica Background

In 1996 several first generation object-oriented mathematical modeling languages and simulation systems (ObjectMath [9], Dymola [6] [7], Omola [16], NMF [22], gPROMS [2], Allan, Smile, etc.) had been developed.

However, the situation with a number of different incompatible object-oriented modeling and simulation languages was not satisfactory. Therefore, in the fall of 1996, a group of researchers from universities and industry started work towards standardization and making this object-oriented modeling technology widely available. This language is called Modelica [10][17][23] and designed primarily for modeling dynamic behavior of engineering systems; moreover, meta-modeling extensions have recently being developed [11]. The language is intended to become a de facto standard.

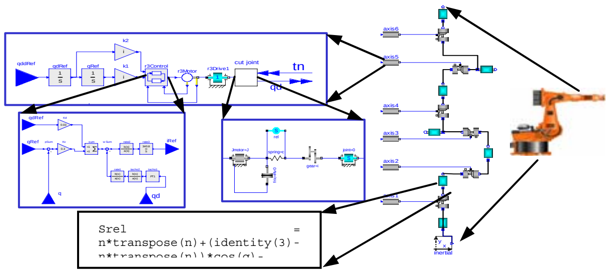
The language allows defining models in a declarative manner, modularly and hierarchically and combining various formalisms expressible in the more general Modelica formalism. The multidomain capability of Modelica allows combining electrical, mechanical, hydraulic, thermodynamic, etc., model components within the same application model.

Compared to most widespread simulation languages available today this language offers several important advantages:

- *Object-oriented mathematical modeling*. This technique makes it possible to create model components, which are employed to support hierarchical structuring, reuse, and evolution of large and complex models covering multiple technology domains.
- *Acausal modeling*. Modeling is based on equations instead of assignment statements as in traditional input/output block abstractions. Direct use of equations significantly increases re-usability of model components, since components adapt to the data flow context in which they are used. Interfacing with traditional software is also available in Modelica.
- *Physical modeling of multiple application domains*. Model components can correspond to physical objects in the real world, in contrast to established techniques that require conversion to “signal” blocks with fixed input/output causality. In Modelica the structure of the model naturally correspond to the structure of the physical system in contrast to block-oriented modeling tools such as Simulink. For application engineers, such “physical” components are particularly easy to combine into simulation models using a graphical editor.
- *A general type system* that unifies object-orientation, multiple inheritance, components/connectors, and templates/generics within a single class construct.

Hierarchical system architectures can easily be described with Modelica thanks to its powerful component model. Components are connected via the connection mechanism realized by the Modelica system, which can be visualized in connection diagrams. The component framework realizes components and connections, and ensures that communication works over the connections.

For systems composed of acausal components with behavior specified by equations, the direction of data flow, i.e., the causality is initially unspecified for connections between those components. Instead the causality is automatically deduced by the compiler when needed. Components have well-defined interfaces consisting of ports, also known as connectors, to the external world. A component may internally consist of other connected components, i.e., hierarchical modeling.. Fig. 1 shows hierarchical component-based modeling of an industry robot.



```
Srel                        =
n*transpose(n)+(identity(3)-
n*transpose(n))*cos(q)-
```

Fig. 1. Hierarchical model of an industrial robot, including components such as motors, bearings, control software, etc. At the lowest (class) level, equations are typically found.

The language design and model library development has proceeded through a number of design meetings (the 52:th May 2007), a nonprofit Modelica Association was started in Linköping, Sweden year 2000, and a conference series was started the same year, with the 5th conference in Vienna Sept. 2006. Several commercial implementations of Modelica (i.e., subsets thereof) are available, including Dymola [5], MathModelica [14], and IDA [3]. This paper primarily focuses on the OpenModelica, which currently is the major Modelica open-source tool effort.

2 The OpenModelica Environment

The OpenModelica environment described in this paper has several goals, including, but not limited to the following:

- Providing an efficient interactive computational environment for the Modelica language.
- Development of a complete reference implementation of Modelica in an extended version of Modelica itself.
- Providing an environment for teaching modeling and simulation. It turns out that with support of appropriate tools and libraries, Modelica is very well suited as a computational language for development and execution of both low level and high level numerical algorithms, e.g. for control system design, solving nonlinear equation systems, or to develop optimization algorithms for complex applications.
- *Language design*, e.g. to further *extend the scope* of the language, e.g. for use in diagnosis, structural analysis, system identification, etc., as well as modeling problems that require extensions such as partial differential equations, enlarged scope for discrete modeling and simulation, etc.
- *Language design* to *improve abstract properties* such as expressiveness, orthogonality, declarativity, reuse, configurability, architectural properties, etc.
- *Improved implementation techniques*, e.g. to enhance the performance of compiled Modelica code by generating code for parallel hardware.
- *Improved debugging* support for equation based languages such as Modelica, for improved ease of use.
- *Easy-to-use* specialized high-level (graphical) *user interfaces* for certain application domains.
- *Visualization* and animation techniques for interpretation and presentation of results.
- *Application usage* and model library development by researchers in various application areas.

In this paper we briefly present a few of the subsystems, as well as some architectural aspects of the environment. Further, we will give examples of the usage of the interactive session handler, the DrModelica notebook, and the debugging support.

2.1 Environment Overview

The OpenModelica environment consists of several interconnected subsystems, as depicted in Fig. 2.

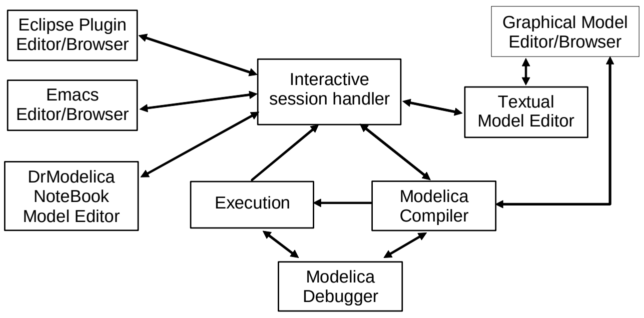

Eclipse Plugin Editor/Browser
Graphical Model Editor/Browser
Interactive session handler
Emacs Editor/Browser
Textual Model Editor
DrModelica NoteBook Model Editor
Execution
Modelica Compiler
Modelica Debugger

Fig. 2. The architecture of the OpenModelica environment.

Arrows denote data and control flow. Several subsystems provide different forms of browsing and textual editing of Modelica code. The debugger currently provides debugging of an extended algorithmic subset of Modelica. The graphical model editor is not really part of OpenModelica but integrated into the system and available from MathCore [14] without cost for academic usage. The following subsystems are currently integrated in the OpenModelica environment:

- *An interactive session handler*, that parses and interprets commands and Modelica expressions for evaluation, simulation, plotting, etc. The session handler also contains simple history facilities, and completion of file names and certain identifiers in commands.
- *A Modelica compiler subsystem*, translating Modelica to C code, with a symbol table containing definitions of classes, functions, and variables. Such definitions can be predefined, user-defined, or obtained from libraries. The compiler also includes a Modelica interpreter for interactive usage and constant expression evaluation. The subsystem also includes facilities for building simulation executables

linked with selected numerical ODE or DAE solvers.

- *An execution and run-time module*. This module currently executes compiled binary code from translated expressions and functions, as well as simulation code from equation based models, linked with numerical solvers. Limited event handling facilities are included for the discrete and hybrid parts of the Modelica language.
- *Emacs textual model editor/browser*. In principle any text editor could be used. We have so far primarily employed Gnu Emacs, which has the advantage of being programmable for future extensions. A Gnu Emacs mode for Modelica has previously been developed. The Emacs mode hides Modelica graphical annotations during editing, which otherwise clutters the code and makes it hard to read. The Emacs mode has been largely superceeded by the Eclipse plugin described below.
- *Eclipse plugin editor/browser/compilation manager*. The Eclipse plugin [21] provides file and class hierarchy browsing and text editing capabilities. Some syntax highlighting facilities are also included. The Eclipse framework has the advantage of making it easier to add future extensions such as refactoring and cross referencing support. A compilation manager is also included. Automatic indentation and debugging facilities are recently being added [20].
- *DrModelica notebook textual model editor*. This subsystem provides a lightweight notebook editor, compared to the more advanced Mathematica notebooks available in MathModelica. This basic functionality still allows essentially the whole DrModelica tutorial to be handled. Hierarchical text documents with chapters and sections can be represented and edited, including basic formatting. Cells can contain ordinary text, graphics, or Modelica models and expressions, which can be evaluated and simulated. However, no mathematical typesetting facilities are yet available in the cells of this notebook editor.
- *Graphical model editor/browser*. This is a graphical connection editor, for component based model design by connecting instances of Modelica classes, and browsing Modelica model libraries for reading and picking component models. The graphic model editor is not really part of OpenModelica but a MathModelica Lite version of the editor, see Section 9, is integrated with the system and provided by MathCore AB [14] without cost for academic usage. The graphic model editor also includes a textual editor for editing model class definitions, and a window for interactive Modelica command evaluation.
- *Modelica debugger*. The current implementation of the debugger [18] provides debugging for an extended algorithmic subset of Modelica, excluding equation-based models and some other features, but including some meta-programming and model transformation extensions [11] to Modelica. This is conventional full-feature debugger, using Emacs for displaying the source code during stepping, setting breakpoints, etc. Various back-trace and inspection commands are available. The debugger also includes a data-view browser for browsing hierarchical data such as tree- or list structures in extended Modelica. As just mentioned, this debugger has been integrated in the OpenModelica Eclipse plugin.

2.2 Implementation Status

The current version of the OpenModelica environment (June 2007) allows most of the expression, equation, algorithm, and function parts of Modelica to be executed interactively, as well as to being compiled into efficient C code. The generated C code is combined with a library of utility functions, a run-time library, and a numerical DAE solver. An external function numeric library interfacing a LAPACK subset and other basic algorithms has also recently been developed.

Not all subsystems are yet integrated as well as is indicated in Fig. 3. Currently there are two versions of the Modelica compiler, one which supports most of standard Modelica including simulation, and is connected to the interactive session handler, the notebook editor, and the graphic model editor, and another meta-programming Modelica compiler version which is integrated with the debugger and Emacs, supports meta-programming Modelica extensions [11], but does not allow equation-based modeling and simulation. Those two versions are currently being merged into a single OpenModelica compiler version.

3 The OpenModelica Client-Server Architecture

The OpenModelica client-server architecture is schematically depicted in Fig. 3, showing three typical clients.

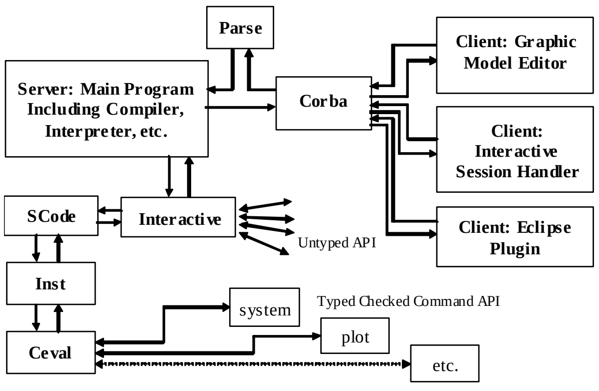

Parse
Server: Main Program Including Compiler, Interpreter, etc.
Corba
Client: Graphic Model Editor
Client: Interactive Session Handler
Client: Eclipse Plugin
SCode
Interactive
Untyped API
Inst
system
Typed Checked Command API
plot
Ceval
etc.

Fig. 3. Client-Server interconnection structure of the compiler/interpreter main program and interactive tool interfaces.

The three clients are: a graphic model editor, an interactive session handler for command interpretation, and the MDT Eclipse plugin.

Commands or Modelica expressions are sent as text from the clients via the CORBA interface, parsed, and divided into two groups by the main program:

- All kinds of declarations of classes, types, functions, constants, etc., as well as equations and assignment statements. Moreover, function calls to the untyped API also belong to this group – a function name is checked if it belongs to the API names. The `Interactive` module handles this group of declarations and untyped API commands.
- Expressions and type checked API commands, which are handled by the `Ceval` module.

The reason the untyped API calls are not passed via `SCode` (a module generating an intermediate form of the abstract syntax tree) and `Inst` (which performs symbolic instantiation of components) to `Ceval` is that `Ceval` can only handle typed calls – the type is always computed and checked, whereas the untyped API prioritizes performance and typing flexibility. The `Main` module checks the name of a called function name to determine if it belongs to the untyped API, and should be routed to `Interactive`.

Moreover, the `Interactive` module maintains an environment of all interactively given declarations and assignments at the top-level, which is the reason such items need to be handled by the `Interactive` module.

4 Simplified Overall Structure of the Compiler

The OpenModelica compiler is divided into a number of modules, to separate different stages of the translation, and to make it more manageable. The top level function is called `main`, and appears as follows in simplified form that emits flat Modelica (leaving out the code generation and symbolic equation manipulation):

```
function main
  input String f  "file name";
protected
  Absyn  ast;
  SCode  scode1;
  SCode  scode2;
algorithm
  ast := Parser.parse(f);
  scode1 := SCode.elaborate(ast);
  scode2 := Inst.elaborate(scode1);
  DAE.dump(scode2);
end main;
```

The simplified overall structure of the OpenModelica compiler is depicted in Fig. 4, showing the most important modules, some of which can be recognized from the above `main` function. The total system contains approximately 40 modules.

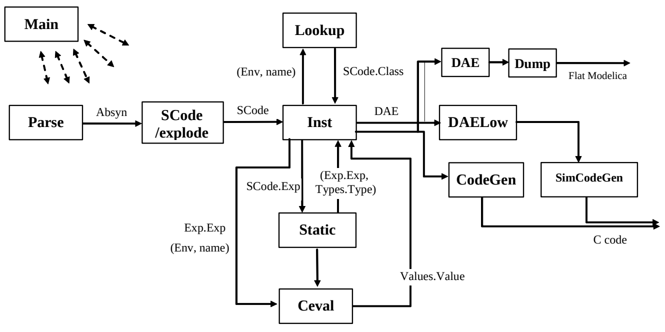


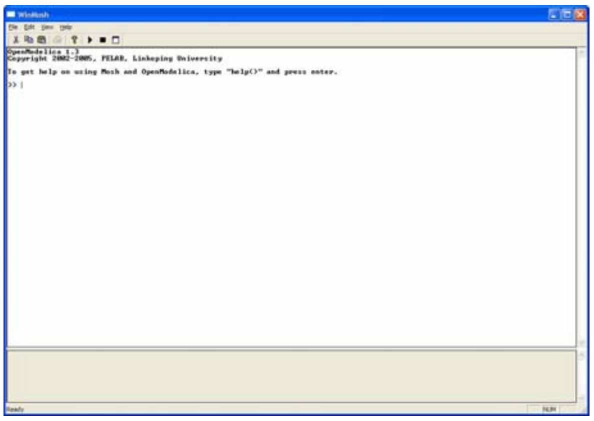
Fig. 4. The OpenModelica compiler (omc) decomposed into modules and data flow connections.

The parser generates abstract syntax (`Absyn`) which is converted to the simplified (`SCode`) intermediate form. The code instantiation module (`Inst`) calls `Lookup` to find a name in an environment. It also generates the DAE equation representation which is simplified by `DAELow`. The `Ceval` module performs compile-time or interactive expression evaluation and returns values. The `Static` module performs static semantics and type checking. The `DAELow` module performs BLT sorting and index reduction (see Chapter 18 in [10]). The DAE module internally uses `Exp.Exp`, `Types.Type` and `Algorithm.Algorithm`; the `SCode` module internally uses `Absyn`.

5 Interactive Session with Examples

The following is an interactive session using the interactive session handler in the OpenModelica environment. (Called OMShell – the OpenModelica Shell).

The Windows version which at installation is made available in the start menu as `OpenModelica->OpenModelica Shell` responds with an interaction window shown in Fig. 5.



```
OpenModelica 1.3
Copyright 2002-2005, PELAB, Linkoping University

To get help on using Mosh and OpenModelica, type "help()" and press enter.

>> |
```

Fig. 5. Initial screen of the interactive session handler.

We enter an assignment of a vector expression, created by the range construction expression `1:12`, to be stored in the variable `x`. The value of the expression is returned.

```
>> x := 1:12
 {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
```

Look at the type of `x`:

```
>> typeOf(x)
"Integer[]"
```

The function `bubblesort` is called to sort this vector in descending order. The sorted result is returned together with its type. Note that the result vector is of type `Real[:]`, instantiated as `Real[12]`, since this is the declared type of the function result. The input `Integer` vector was automatically converted to a `Real` vector according to the Modelica type coercion rules. The function is automatically compiled when called if this has not been done before.

```
>> bubblesort(x)
{12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1}
```

It is also possible to give operating system commands via the `system` utility function. A command is provided as a string argument. The example below shows the `system` utility applied to the UNIX command `cat`, which here outputs the contents of the file `bubblesort.mo` to the output stream. However, the cat command does not boldface Modelica keywords – this improvement has been done by hand for readability.

```
>> cd("C:/OpenModelica1.4.0/testmodels")
>> system("cat bubblesort.mo")

function bubblesort
  input Real[:] x;
  output Real[size(x,1)] y;
protected
  Real t;
algorithm
  y := x;
  for i in 1:size(x,1) loop
    for j in 1:size(x,1) loop
      if y[i] > y[j] then
            t := y[i];
            y[i] := y[j];
            y[j] := t;
      end if;
    end for;
  end for;
end bubblesort;
```

It is also possible to enter a function directly into the session handler.

```
>> function MySqr input Real x; output
Real y; algorithm y:=x*x; end MySqr;
Ok
```

And then call the function:

```
>> b:=MySqr(2)
4.0
```

Another built-in command is `cd`, the *change current directory* command. The resulting current directory is returned as a string.

```
>> cd("..")
"/home/petfr/modelica"
```

We load a model, here the whole Modelica standard library:

```
>> loadModel(Modelica)
true
```

We also load a file containing the `dcmotor` model:

```
>>
loadFile("C:/OpenModelica1.4.0/testmodels
/dcmotor.mo")
true
```

It is simulated:

```
>> simulate(dcmotor,startTime=0.0,
stopTime=10.0)

record
    resultFile = "dcmotor_res.plt"
end record
```

We list the source code of the model:

```
>> list(dcmotor)
"model dcmotor
Modelica.Electrical.Analog.Basic.Resistor
r1(R=10);
Modelica.Electrical.Analog.Basic.Inductor
i1;
Modelica.Electrical.Analog.Basic.EMF emf1;
Modelica.Mechanics.Rotational.Inertia
load;
Modelica.Electrical.Analog.Basic.Ground g;
Modelica.Electrical.Analog.Sources.Constan
tVoltage v;
equation
  connect(v.p,r1.p);
  connect(v.n,g.p);
  connect(r1.n,i1.p);
  connect(i1.n,emf1.p);
  connect(emf1.n,g.p);
  connect(emf1.flange_b,load.flange_a);
end dcmotor;
```

We plot part of the simulated result:

```
>> plot({load.w,load.phi})
true
```

The output is shown in Fig. 6.

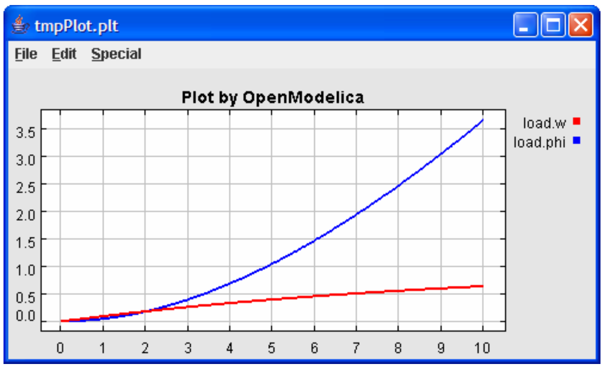


Fig. 6. Plot of the simulated dcmotor model.

Clear all loaded libraries and models:

```
>> clear()
true
```

List the loaded models – but nothing left:

```
>> list()
""
```

We load another model, the `Influenza` model:

```
>>
loadFile("M:/modeq/VC7/Setup/testmodels/
Influenza.mo")
true
```

It is simulated:

```
>> simulate(Influenza,startTime=0.0,
stopTime=3.0)
record
    resultFile = "Influenza_res.plt"
end record
```

The simulated population is plotted, which is shown in Fig. 7.

```
>> plot({Infected_Popul.p})
true
```

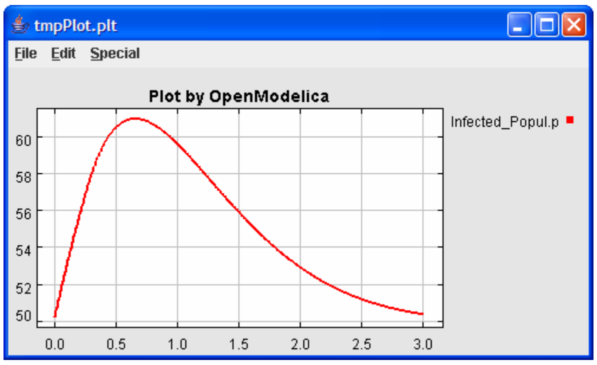


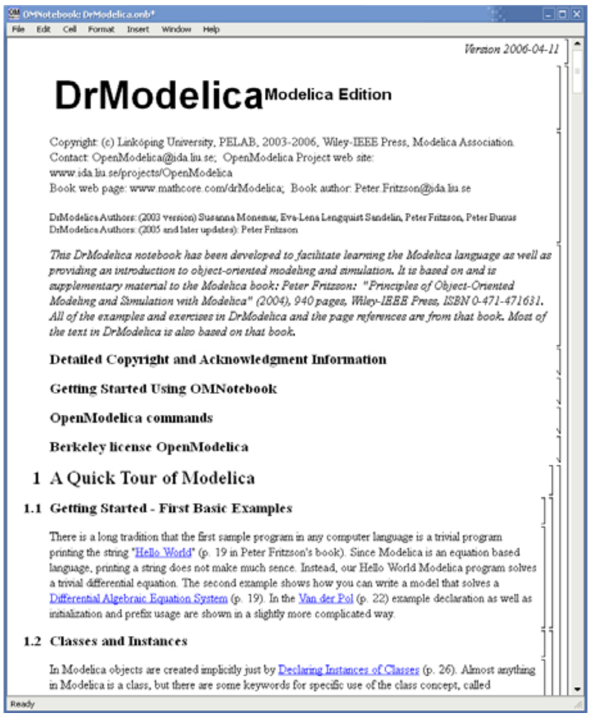
Fig. 7. Plot of the Influenza model.

For a complete list of the available commands, see [13].

6 DrModelica Notebook and Textual Model Editor

The OpenModelica electronic notebook (OMNotebook) and model editor subsystem [1] [8] can be used as a textual modeling interface for Modelica, or as a Modelica tutoring system, i.e., a simplified version of the earlier Mathematica-based DrModelica tutoring system for teaching Modelica.

The simplified OpenModelica electronic notebooks are however able to handle the full DrModelica tutorial material, containing most of the Modelica examples in [10]. It is advanced enough to represent hierarchical documents, simple type setting, text editing, graphic cells, etc.


OMNotebook: DrModelica.onb*
File Edit Cell Format Insert Window Help
Ready

Version 2006-04-11

DrModelica **Modelica Edition**

Copyright (c) Linköping University, PELAB, 2003-2006, Wiley-IEEE Press, Modelica Association.
Contact: OpenModelica@ida.liu.se; OpenModelica Project web site:
www.ida.liu.se/projects/OpenModelica
Book web page: www.mathcore.com/drModelica; Book author: Peter.Fritzson@ida.liu.se

DrModelica Authors: (2003 version) Susanna Monemar, Eva-Lena Lengquist Sandelin, Peter Fritzson, Peter Bunus
DrModelica Authors: (2005 and later updates): Peter Fritzson

This DrModelica notebook has been developed to facilitate learning the Modelica language as well as providing an introduction to object-oriented modeling and simulation. It is based on and is supplementary material to the Modelica book: Peter Fritzson: "Principles of Object-Oriented Modeling and Simulation with Modelica" (2004), 940 pages, Wiley-IEEE Press, ISBN 0-471-471631. All of the examples and exercises in DrModelica and the page references are from that book. Most of the text in DrModelica is also based on that book.

Detailed Copyright and Acknowledgment Information

Getting Started Using OMNotebook

OpenModelica commands

Berkeley license OpenModelica

1 A Quick Tour of Modelica

1.1 Getting Started - First Basic Examples

There is a long tradition that the first sample program in any computer language is a trivial program printing the string "Hello World" (p. 19 in Peter Fritzson's book). Since Modelica is an equation based language, printing a string does not make much sence. Instead, our Hello World Modelica program solves a trivial differential equation. The second example shows how you can write a model that solves a Differential Algebraic Equation System (p. 19). In the Van der Pol (p. 22) example declaration as well as initialization and prefix usage are shown in a slightly more complicated way.

1.2 Classes and Instances

In Modelica objects are created implicitly just by Declaring Instances of Classes (p. 26). Almost anything in Modelica is a class, but there are some keywords for specific use of the class concept, called

Fig. 8. The start page (main page) of DrModelica in the OpenModelica notebook system.

This is exemplified by Fig. 8, showing the DrModelica main page (start page) in the teaching material.

6.1 Some OpenModelica Notebook Commands

The current prototype of OpenModelica notebooks includes, but is not limited, to the following operations:

- Opening and closing groups of cells by double clicking the hierarchical tree view (to the right).
- Evaluation of Modelica code, commands, and expressions in input cells by typing `SHIFT+RETURN`. The evaluation results are shown in a created output cell.
- Opening loading, and saving notebook files in XML (.onb) format.
- Terminating the notebook subsystem (`ALT+Q` or `ALT+F4`).
- Select a cell, by a single click on the cell in the tree view to the right.
- Possibility to edit the style template to change the appearance of different cell types.
- Move cursor, by `CTRL + UP ARROW` or `CTRL + DOWN ARROW`.
- Close current document (`CTRL+W`).
- Select and copy text inside a cell.

7 Modelica Algorithmic Subset Debugger

This section presents a comprehensive Modelica debugger [18] for an extended algorithmic subset of the Modelica language. The debugger replaces debugging of algorithmic code using primitive means such as print statements or asserts which is complex, time-consuming and error- prone.

Two versions of the debugger has been developed. The first version, [18], is based on Emacs as user interface. The second more recent and substantially improved version is integrated in Eclipse as part of the Open-Modelica MDT Eclipse plugin, and is being released at the time of this writing. Some aspects of this new debugger are described in [19].

The debugger is portable since it is based on transparent source code instrumentation techniques that are independent of the implementation platform.

The usual debugging functionality found in debuggers for procedural or traditional object-oriented languages is supported, such as setting and removing breakpoints, single-stepping, inspecting variables, back-trace of stack contents, tracing, etc.

In this section we present parts of the Emacs version of the debugger functionality. Some of the functionality of the Emacs version of the debugger is shown Fig. 9.

7.1 Debugger Commands

The Emacs Modelica debug mode is implemented as a specialization of the Grand Unified Debugger (GUD) interface (`gud-mode`) from Emacs. Because the Modelica debug mode is based on the GUD interface, some of the commands have the same familiar key bindings.

The actual commands sent to the debugger are also presented together with GUD commands preceded by the Modelica debugger prompt: `mdb@>`.

If the debugger commands have several alternatives these are presented using the notation:

`alternative1|alternative2|....`

The optional command components are presented using notation: `[optional]`.

In the Emacs interface: `M-x` stands for holding down the `Meta` key (mapped to `Alt` in general) and pressing the key after the dash, here `x`, `C-x` stands for holding down the `Control (Ctrl)` key and pressing `x`, `<RET>` is equivalent to pressing the `Enter` key, and `<SPC>` to pressing the `Space` key.

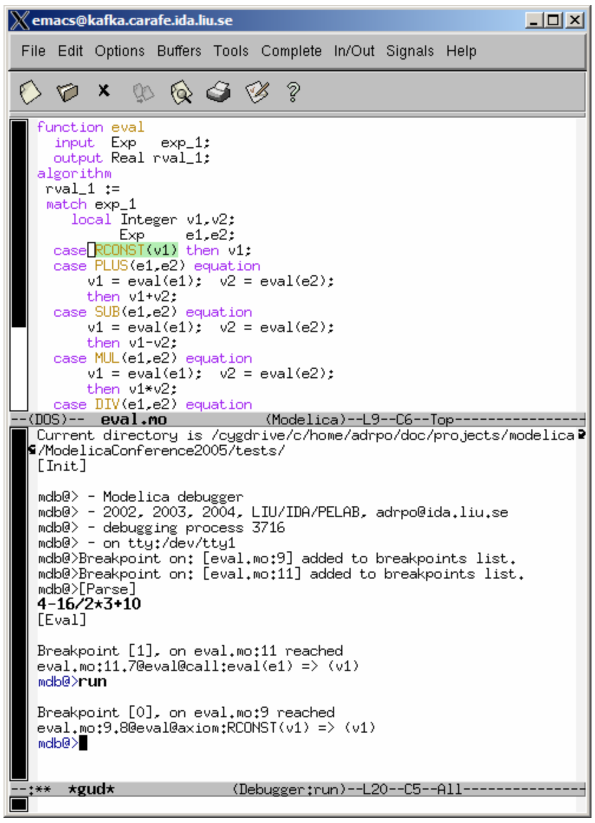
7.2 Starting the Modelica Debugging Subprocess

The command for starting the Modelica debugger under Emacs is the following:

```
M-x modelicadebug <RET> executable
<RET>
```

7.3 Setting/Deleting Breakpoints

A part of a session using this type of commands is shown in Fig. 9 below.


emacs@kafka.carafe.ida.liu.se
File Edit Options Buffers Tools Complete In/Out Signals Help

```
function eval
  input  Exp   exp_1;
  output Real rval_1;
algorithm
 rval_1 :=
 match exp_1
    local Integer v1,v2;
          Exp     e1,e2;
  case RCONST(v1) then v1;
  case PLUS(e1,e2) equation
      v1 = eval(e1);  v2 = eval(e2);
      then v1+v2;
  case SUB(e1,e2) equation
      v1 = eval(e1);  v2 = eval(e2);
      then v1-v2;
  case MUL(e1,e2) equation
      v1 = eval(e1);  v2 = eval(e2);
      then v1*v2;
  case DIV(e1,e2) equation
--(DOS)--  eval.mo           (Modelica)--L9--C6--Top---------------
Current directory is /cygdrive/c/home/adrpo/doc/projects/modelica
/ModelicaConference2005/tests/
[Init]

mdb@> - Modelica debugger
mdb@> - 2002, 2003, 2004, LIU/IDA/PELAB, adrpo@ida.liu.se
mdb@> - debugging process 3716
mdb@> - on tty:/dev/tty1
mdb@>Breakpoint on: [eval.mo:9] added to breakpoints list.
mdb@>Breakpoint on: [eval.mo:11] added to breakpoints list.
mdb@>[Parse]
4-16/2*3+10
[Eval]

Breakpoint [1], on eval.mo:11 reached
eval.mo:11.7@eval@call:eval(e1) => (v1)
mdb@>run

Breakpoint [0], on eval.mo:9 reached
eval.mo:9.8@eval@axiom:RCONST(v1) => (v1)
mdb@>
--:**  *gud*           (Debugger:run)--L20--C5--All--------------
```

Fig. 9. Using breakpoints.

This is only a brief presentation of a subset of the debugger functionality. See the OpenModelica Users Guide [13] for a more complete description.

8 Modelica Development Tooling (MDT) Eclipse Plug-In

The Modelica Development Tooling (MDT) Eclipse Plugin [21] integrates the OpenModelica compiler with Eclipse. MDT, together with the OpenModelica compiler, provides an environment for working with Modelica development projects.

The following features are available:

- Browsing support for Modelica projects, packages, and classes
- Wizards for creating Modelica projects, packages, and classes
- Syntax color highlighting
- Syntax checking
- Code completion
- Automatic indentation
- Automatic display of information about declared items
- Browsing of Modelica modules and libraries

8.1 Using the Modelica Perspective

The most convenient way to work with Modelica projects is to use to the Modelica perspective. To switch to the Modelica perspective, choose the `Window` menu item, pick `Open Perspective` followed by

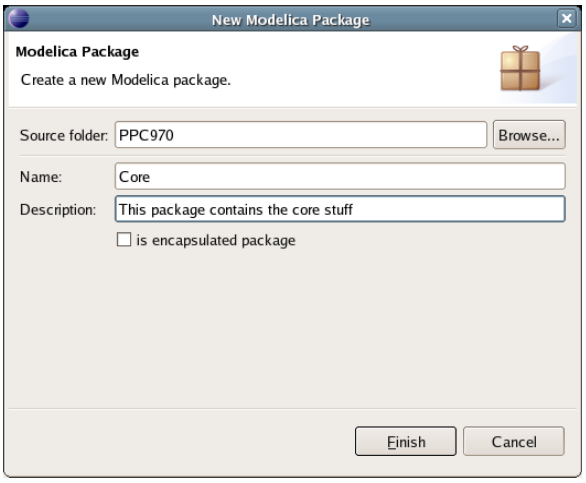
`Other...` Select the `Modelica` option from the dialog presented and click `OK`.

8.2 Creating a Project

To start a new project, use the `New Modelica Project` Wizard. It is accessible through `File->New->Modelica Project` or by right-clicking in the Modelica Projects view and selecting `New->Modelica Project`.

8.3 Creating a Package

To create a new package inside a Modelica project, select `File->New->Modelica Package`. Enter the desired name of the package and a description of what it contains.


New Modelica Package

Modelica Package

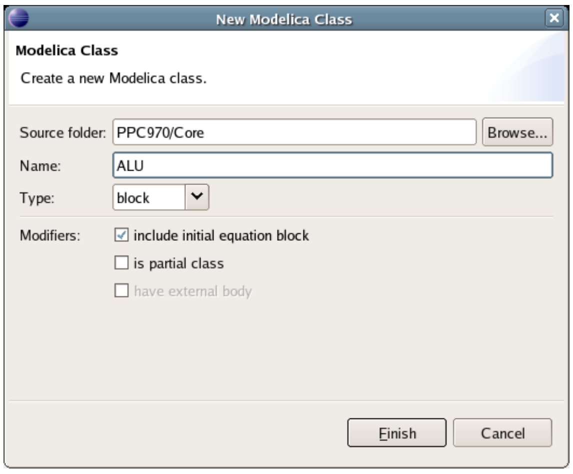
Create a new Modelica package.

Source folder:	PPC970
Name:	Core
Description:	This package contains the core stuff
	☐ is encapsulated package

Fig. 10. Creating a new package.

8.4 Creating a Class

To create a new Modelica class, select where in the hierarchy that you want to add your new class and select `File->New->Modelica Class`. When creating a Modelica class you can add different restrictions on what the class can contain. These can for example be `model`, `connector`, `block`, `record`, or `function`.


New Modelica Class

Modelica Class

Create a new Modelica class.

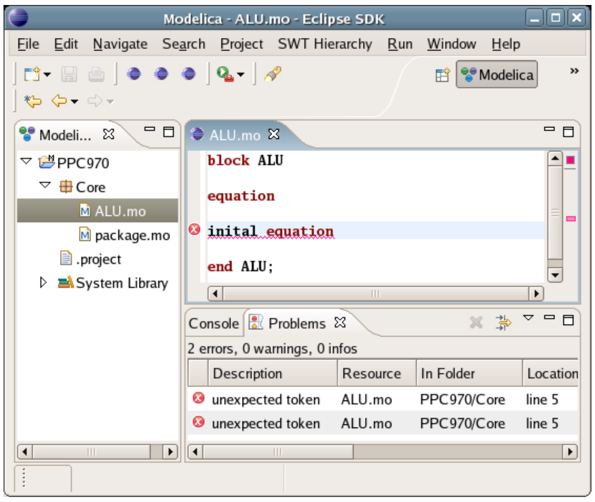
Source folder:	PPC970/Core
Name:	ALU
Type:	block
Modifiers:	☑ include initial equation block
	☐ is partial class
	☐ have external body

Fig. 11. Creating a new class.

When you have selected your desired class type, you can select modifiers that add code blocks to the generated code. '`Include initial code block`' will for example add the line '`initial equation`' to the class.

8.5 Syntax Checking

Whenever a Modelica (`.mo`) file is saved by the Modelica Editor, it is checked for syntactical errors. Any errors that are found are added to the Problems view and also marked in the source code editor. Errors are marked in the editor as a red circle with a white cross, a squiggly red line under the problematic construct, and as a red marker in the right-hand side of the editor. If you want to reach the problem, you can either click the item in the Problems view or select the red box in the right-hand side of the editor.


Modelica - ALU.mo - Eclipse SDK

```
block ALU

equation

inital equation

end ALU;
```

2 errors, 0 warnings, 0 infos

Description	Resource	In Folder	Location
unexpected token	ALU.mo	PPC970/Core	line 5
unexpected token	ALU.mo	PPC970/Core	line 5

Fig. 12. Syntax checking.

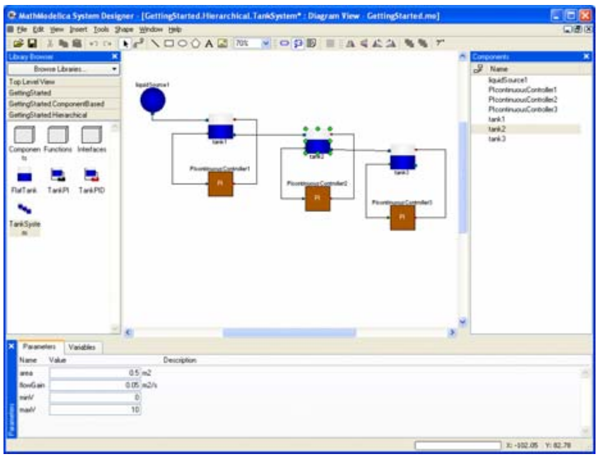
9 Graphic Editing with MathModelica Lite

A model can be built using the graphical model editor by using drag-and-drop of already developed and freely available model components from the Modelica Standard Library.

The Modelica Standard Library can be loaded into the OpenModelica environment when the MathModelica Lite model editor is started and can be browsed using the class browser visible at the left of Fig. 13 below.

This section just gives a short sample of using the graphical model editor. See `www.mathcore.com` for the complete MathModelica System Designer User's Guide, which includes additional capabilities in modeling, simulation, plotting, and Modelica library support.

As mentioned previously, there is no graphical model editor in OpenModelica, but the Lite edition of the MathModelica model editor from MathCore that works together with OpenModelica can be downloaded from the OpenModelica web site or directly from `www.mathcore.com/products/math-modelica/lite/`. The Lite edition of the editor is free for academic non-commercial usage.



Name	Value	Description
area	0.5	m2
flowGain	0.05	m2/s
minV	0	
maxV	10	

Fig. 13. The model editor of MathModelica Lite with the class browser to the left, the graphic editing area in the middle, and the instance component browser to the right.

10 Conclusion

We have presented some aspects of the OpenModelica environment, including facilities for modeling, simulation, and debugging Modelica code. A number of objectives of the OpenModelica environment have been presented and some examples illustrated. It has been demonstrated that the OpenModelica environment includes many valuable features for engineers and researchers, and it is the only Modelica environment so far with good support for debugging Modelica algorithmic code as well as support for meta-programming integrated in the language. We believe that this open source platform can be part of forming the foundation of the next generation of the Modelica language and environment development efforts, both from a research perspective and a system engineering usage point of view.

11 Acknowledgments

This work was supported by SSF in the VISIMOD project, by Vinnova in SafeModSim project, by Vetenskapsrådet, by Linköping University, and by MathCore Engineering AB. Thanks to Ingemar Axelsson and Anders Fernström for design and implementation regarding the OMNotebook, and to Andreas Remar and Elmir Jagudin for design and implementation regarding the MDT Eclipse plugin.

Note: Most of the material presented in this paper has been previously published, especially as part of [12] and [10].

12 References

[1] Ingemar Axelsson. OpenModelica Notebook for Interactive Structured Modelica Documents. Master Thesis LITH-IDA-EX-05/080-SE, 2005.

[2] Paul Barton and Costas Pantelides. The Modelling of Combined Discrete/Continuous Processes. AIChemE Journal, 40, pp. 996–979, 1994.

[3] Equa AB. The IDA simulation tool. www.equa.se. [Accessed 2006].

[4] Ernst Christen and Kenneth Bakalar. VHDL-AMS—A Hardware Description Language for Analog and Mixed-Signal Applications. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing. Vol. 46, Issue 10, pp. 1263–1272, Oct. 1999.

[5] Dynasim AB. Dymola—Dynamic Modeling Laboratory with Modelica, Users Manual, Version 6.0. Dynasim AB, Research Park Ideon, SE-223 70, Lund, Sweden, 2006.

[6] Hilding Elmqvist. A Structured Model Language for Large Continuous Systems. Ph.D. thesis, TFRT-1015, Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978.

[7] Hilding Elmqvist, Dag Bruck, and Martin Otter. Dymola—User's Manual. Dynasim AB, Research Park Ideon, SE-223 70, Lund, Sweden, 1996.

[8] Anders Fernström. Extending OMNotebook – An Interactive Notebook for Structured Modelica Documents. Master thesis, LITH-IDA-EX--06/057—SE, Linköping University, 2006.

[9] Peter Fritzson, Lars Viklund, Dag Fritzson, Johan Herber. High-Level Mathematical Modelling and Programming, IEEE Software, 12(4):77-87, July 1995, http://www.ida.liu.se/labs/pelab/omath

[10] Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, 940 pp., ISBN 0-471-471631, Wiley-IEEE Press, 2004.

[11] Peter Fritzson, Adrian Pop, and Peter Aronsson. Towards Comprehensive Meta-Modeling and Meta-Programming Capabilities in Modelica. In Proceedings of the 4th International Modelica Conference, Hamburg, Germany, March 7-8, 2005.

[12] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, David Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. In Simulation News Europe (SNE), 44, January 2006. See also www.ida.liu.se/projects/OpenModelica.

[13] Peter Fritzson et al. OpenModelica Users Guide and OpenModelica System Documentation, May 2006. www.ida.liu.se/projects/ OpenModelica

[14] MathCore Engineering AB. MathModelica User's Guide. 2006. www.mathcore.com, 2006.

[15] MathWorks. The Mathworks - Simulink - Simulation and Model-Based Design. http://www.

mathworks.com/products/simulink/ [Last accessed: 15 May 2006].

[16] Sven-Erik Mattsson and Mats Andersson. The Ideas Behind Omola. In Proceedings of the 1992 IEEE Symposium on Computer-Aided Control System Design (CADCS ’92), Napa, California, Mar. 1992.

[17] Modelica Association. The Modelica Language Specification Version 2.2, March 2005. http://www.modelica.org.

[18] Adrian Pop and Peter Fritzson: A Portable Debugger for Algorithmic Modelica Code. In Proceedings of the 4th International Modelica Conference, Hamburg, Germany, March 7-8, 2005.

[19] Adrian Pop and Peter Fritzson. Run-time Debugging of Equation-based Object-oriented Languages. Submitted to SIMS'2007, Gothenburg, Sweden, Oct 2007.

[20] Adrian Pop, Peter Fritzson, Andreas Remar, Elmir Jagudin, David Akhvlediani. OpenModelica Development Environment with Eclipse Integration for Browsing, Modeling, and Debugging. In Proc. of Modelica’2006, the 5th Int. Modelica Conf., Vienna, Sept 4-5, 2006.

[21] Andreas Remar and Elmir Jagudin. Modelica Development Tooling for Eclipse. Master Thesis LITH-IDA-EX–06/024–SE, April 10, 2006.

[22] Per Sahlin. Modelling and Simulation Methods for Modular Continuous Systems in Buildings. Ph.D. thesis, Dept. of Building Science, Royal Inst. of Technology Stockholm, Sweden, May 1996.

[23] Michael Tiller. Introduction to Physical Modeling with Modelica. 366 pages. ISBN 0-7923-7367-7, Kluwer Academic Publishers, 2001.