An Open Source Modelica Graphic Editor Integrated with Electronic Notebooks and Interactive Simulation

Syed Adeel Asghar[1], Sonia Tariq[1], Mohsen Torabzadeh-Tari[1], Peter Fritzson[1], Adrian Pop[1], Martin Sjölund[1], Parham Vasaiely[2], Wladimir Schamai[2]

[1]PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, SE-581 83 Linköping, Sweden
[2]EADS Innovation Works, Engineering & Architecture, 21129 Hamburg, Germany
adeel.asghar@liu.se, x10sonta@ida.liu.se, {mohsen.torabzadeh-tari,peter.fritzson,adrian.pop,martin.sjolund}@liu.se,
Parham.Vasaiely@gmx.de, Wladimir.schamai@eads.net

Abstract

This paper describes the first open source Modelica graphic editor which is integrated with interactive electronic notebooks and online interactive simulation.

The work is motivated by the need for easy-to-use graphic editing of Modelica models using OpenModelica, as well as needs in teaching where the student should be able to interactively modify and simulate models in an electronic book. Models can be both textual and graphical. The interactive online simulation makes the simulation respond in real-time to model changes, which is useful in a number of contexts including immediate feedback to students.

Keywords: Graphic editing, notebook, teaching, interactive, Modelica, modeling, simulation, online

1 Introduction

OMEdit, the OpenModelica Connection Editor, is the new Graphical User Interface for graphic model editing in OpenModelica. It is implemented in C++ using the Qt 4.7 graphical user interface library, and supports the Modelica Standard Library version 3.1 that comes with the OpenModelica installation.

OMEdit provides a user friendly environment for:

- *Modeling* – Easy Modelica model creation.
- *Pre-defined Models* – Browsing the Modelica Standard library to access the provided models.
- *User defined models* – Users can create their own models for immediate usage and later refinement and reuse.
- *Component Interfaces* – Smart connection editing for drawing and editing connections between model interfaces.
- *Simulation subsystem* – Subsystem for running simulations (not online) and specifying simulation parameters start and stop time, etc.
- *Online Simulation* – Online interactive simulation where the simulation responds in real-time to user input and changes to parameters.
- *Plotting* – Interface to plot variables from simulated models.
- *OMNotebook integration* – being able to open a graphical connection diagram in an electronic notebook, edit it, and paste it back.

OMEdit uses the OmniORB CORBA implementation to communicate with the OpenModelica Compiler.

Modelica 3.2 Graphical Annotations are interpreted for drawing Modelica Standard Library component models and user defined models. As a result, the interoperability with other Modelica tool vendors becomes easier as the Modelica icon and diagrams defined in other tools supporting the Modelica 3.1 or Modelica 3.2 standards are easily handled in OMEdit. The annotations are also used for displaying Modelica documentation in OMEdit.

1.1 Structure of the Paper

Section 3 describes the usage of OMEdit and also demonstrates how a `DCmotor` model is created using OMEdit. Section 4 explains the OMEdit communication process with OMC through the CORBA interface. How user defined and Modelica component model shapes are created through annotations is discussed in section 5.

Section 6 elaborates the interactive simulation mechanism that is still under development in OMEdit. Section 7 briefly describes how OMEdit can interact with OMNotebook and how users can launch electronic

notebooks in OMEdit. Moreover, Section 8 presents related work and in the end, Section 9 suggests some future work.

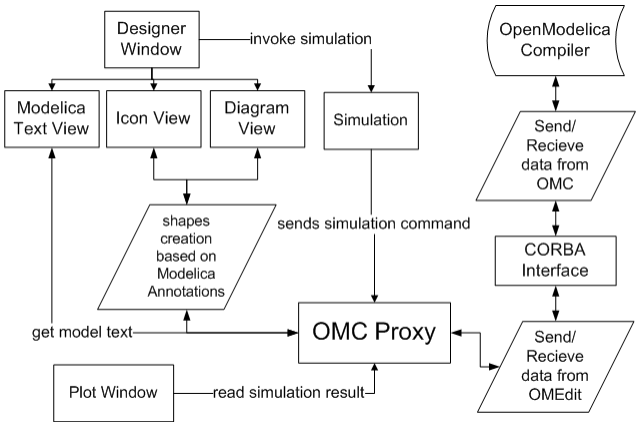


Figure 1. OMEdit High-level View.

2 Requirements and Motivation

This work is motivated by the need for easy-to-use graphic editing of Modelica models using OpenModelica, as well as needs in teaching where the student should be able to interactively modify and simulate textual and graphical models in an electronic notebook.

The interactive online simulation makes the simulation respond in real-time to model changes, which is useful in a number of contexts, especially teaching where immediate feedback to students enhances the effectiveness of learning.

A recently developed interactive learning material called DrControl is depicted in Figure 2.

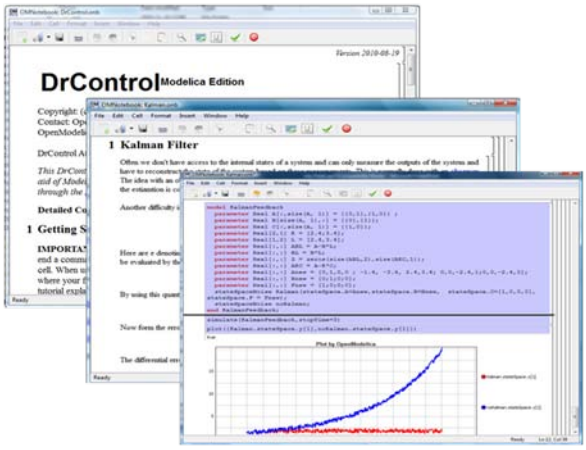


Figure 2. DrControl for teaching control theory with Modelica.

DrControl is a new active electronic notebook course material based on OMNotebook for teaching control theory and modeling with Modelica, including graphic connection diagrams supported by OMEdit. It contains explanations about basic concepts of control theory along with Modelica exercises. Observer models, Kalman filters, and linearization of non-linear problems are some of the topics in the course used in control of a pendulum, a DC motor, and a tank system model among others.

3 Using OMEdit

This section gives a brief introduction about how to use OMEdit and also demonstrates how to create a `DCmotor` model.

3.1 Introductory Model in OMEdit

Since Modelica is an equation-based language and OMEdit is a connection editor, we will for a small introductory model demonstration in OMEdit show how a `DCmotor` model is created in OMEdit.

3.1.1 Creating a new file

Creating a new file/model in OMEdit is rather straightforward. In OMEdit the new file can be of type `model`, `class`, `connector`, `record`, `block`, `function` or `package`. The user can create any of the model types mentioned above by selecting `File > New` from the menu. Alternatively, you can also click on the drop down button beside `new icon` shown in the toolbar right below the File menu. See Figure 3.

In this introductory example we will create a new model named `DCmotor`. By default the newly created model will open up in the tabbed view of OMEdit, also called `Designer Window`, and become visible.

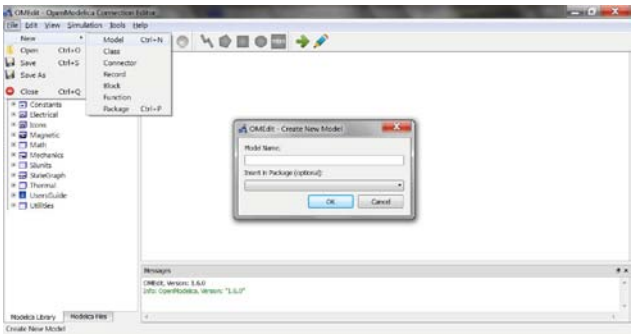


Figure 3. Creating a new file/model.

All the models are created in the OMC global scope unless the user specifies the parent package for it.

3.1.2 Adding Component Models

The Modelica Standard Library is loaded automatically and is available in the left dock window. The library is retrieved through the `loadModel(Modelica)` API function and is loaded into the OMC symbol table and workspace after the command is completed.

Instances of the component models available in the Modelica Standard Library can be added to the currently edited model by doing a drag and drop from the `Library Window`. Navigate to the component model in the library tree, click on it, drag it to the model you are building while pressing the mouse left button, and drop the component where you want to place it in the model.

For this example we will add four components as instances of the models `Ground`, `Resistor`, `Inductor` and `EMF` from the `Modelica.Electrical.Analog.Basic` package, an instance of the model `SignalVoltage` from the `Modelica.Electrical.Analog.Sources` package, one instance of the model `Inertia` from the `Modelica.Mechanics.Rotational.Components` package and one last instance of the model `Step` from the `Modelica.Blocks.Sources` package.

3.1.3 Making Connections

In order to connect one component model to another the user simply clicks on any of the ports. Then it will start displaying a connection line. Then move the mouse to the component where you want to finish the connection and click on the component port where the connection should end. You do not need to hold the mouse left button down for drawing connections.

In order to have a functioning `DCmotor` model, connect the `Resistor` to the `Inductor` and the `SignalVoltage`, `EMF` to `Inductor` and `Inertia`, `Ground` to `SignalVoltage` and `EMF`, and finally `Step` to `SignalVoltage`. Check Figure 4 to see how the `DCmotor` model looks like after connections.

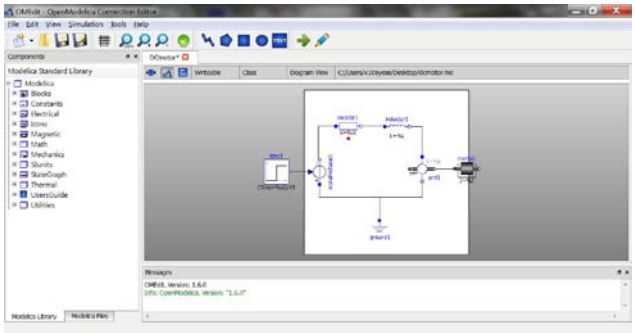


Figure 4. DCmotor model after connections.

3.1.4 Simulating the model

OpenModelica models are simulated using the `simulate` command of OMC. The `simulate` command has following parameters;

- Simulation Interval
  - Start Time
  - Stop Time
- Output Interval
  - Number of Intervals
  - Output Interval
- Integration
  - Method
  - Tolerance
  - Fixed Step Size

The OpenModelica Connection Editor provides an easy interface for simulation of models and allows the user to fill in the parameters before starting the simulation process.

The OMEdit `Simulation` dialog can be launched either from `Simulation > Simulate` or by clicking the `simulate` icon from the toolbar. Once the user clicks on `simulate!` button, OMEdit starts the simulation process, at the end of the simulation process the `Plot Variables` window, useful for plotting, will appear at the right side. Figure 5 shows the simulation dialog.

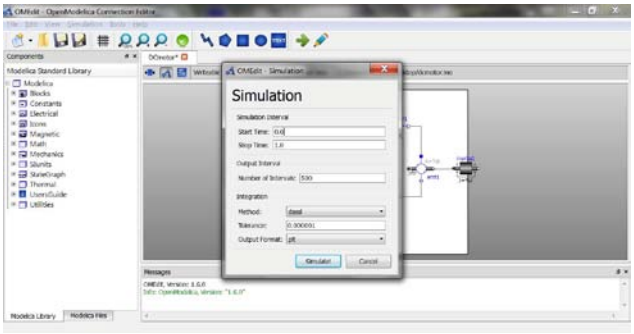


Figure 5. Simulation Dialog.

3.1.5 Plotting Variables from Simulated Models

The instance variables that are candidates for plotting are shown in the right dock window. This window is automatically launched once the user simulates the model; the user can also launch this window manually either from `Simulation > Plot Variables` or by clicking on the `plot` icon from toolbar. It contains the list of variables that are possible to use in an OpenModelica plot. The plot variables window contains a tree structure of variables; there is a checkbox beside each variable. The user can launch the plotted graph window by clicking the checkbox.

Figure 6 shows the complete `DCmotor` model along with the list of plot variables and an example plot window.

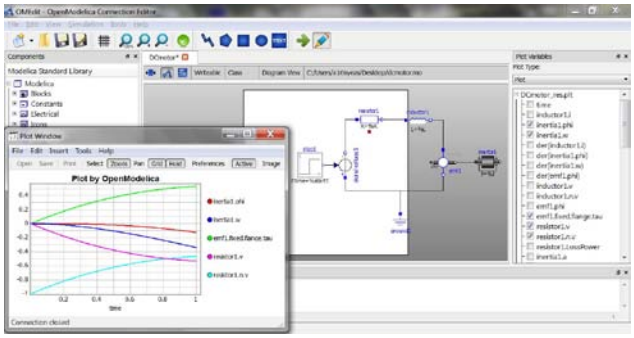


Figure 6. Plotted variables.

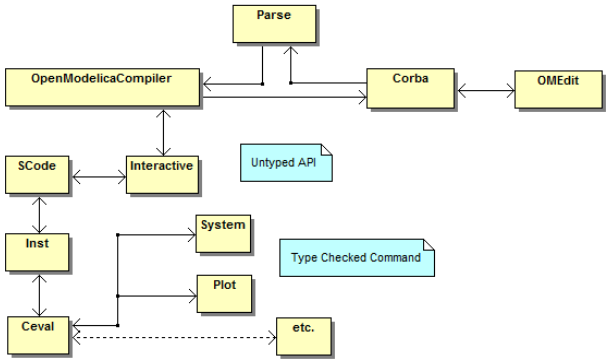


Figure 7. Client-Server interconnection structure of the compiler/interpreter main program and some interactive tool interfaces.

4 Communication with OMC

For graphical modeling OMEdit needs to draw shapes/component models that are defined by Modelica annotations. In order to obtain the Modelica annotations OMEdit must be able to communicate with the OpenModelica Compiler through the CORBA interface.

4.1 OMC CORBA Interface

OMC is a short name for the OpenModelica Compiler. There are two methods to invoke it:

- As a whole program, called at the operating-system level, e.g. as a command.
- As a server, called via a CORBA client-server interface from client applications.

OMEdit uses the second method to invoke the OpenModelica Compiler/Interpreter OMC, since this allows interactive access and querying of the models, needed for interactive graphic editing.

4.2 The CORBA Client Server Architecture

The Figure 7 below describes the design of the OpenModelica client server architecture. OMEdit plays the role of client in this architecture. It sends and receives commands through the CORBA interface. The messages and expressions from the CORBA interface to OMC are divided into two groups. The first group contains the commands which are evaluated by the `Ceval` module and the second group consists of expressions which are handled by the `Interactive` module.

Messages via the CORBA interface are of two kinds. The first group consists of expressions or user commands which are evaluated by the `Ceval` module. The second group consists of declarations of classes, variables, etc., assignments, and client-server API calls that are handled via the `Interactive` module, which also stores information about interactively declared/assigned items at the top-level in an environment [1].

4.3 Invoking OMC through CORBA

In order to communicate with OMC through CORBA you need to start `omc.exe` as a process with special parameters passed to it. The OMC binary executable file is located in `$OPENMODELICAHOME/bin`. OMEdit invokes OMC with a special CORBA flag `+d=interactiveCorba` telling OMC to start with the interactive CORBA communication environment. The complete command will look like this:

```
omc.exe +d=interactiveCorba.
```

OMEdit starts a new OMC process for its each instance. Only one OMC is linked to each instance of OMEdit. However, for some special tasks a new OMC is used and is removed as soon as the task is completed.

OMEdit also passes one special argument flag `+c` to OMC which is used to specify the Interoperable Object Reference (IOR) file name. By default the IOR file is created in the temp directory. OMEdit uses the application session identity number along with the current timestamp to ensure that each instance of OMEdit gets a new OMC.

When OMC is started with the `+d= interactiveCorba` flag, it will create a file named `openmodelica.objid` (name depends on the `+c` argument flag value of OMC) in the temp directory of operating system. This file contains the CORBA IOR.

4.4 What to do with the CORBA IOR File?

The IOR File contains the CORBA object reference as a string. The CORBA object is created by reading the string written in the IOR File. Here is an example with Qt C++ source code for starting OMC from OMedit and creating a CORBA object:

```
// create a unique file name
QString fileIdentifier;fileIdentifier =
qApp-
>sessionId().append(QTime::currentTime().t
oString().remove(":"));

QStringList parameters;
parameters << QString("+c=").append(this-
>mName).append(fileIdentifier) <<
QString("+d=interactiveCorba");

// start the OMC process
QProcess *omcProcess = new QProcess();
omcProcess->start( omcPath, parameters );

// read the file created by omc.exe
QFile objectRefFile (path_to_IOR_File);
int argc = 2;
static const char *argv[] = { "-
ORBgiopMaxMsgSize", "10485760" };
CORBA::ORB_var orb = CORBA::ORB_init(argc,
(char **)argv);
objectRefFile.open(QIODevice::ReadOnly);
char buf[1024];

// read the IOR string
objectRefFile.readLine( buf, sizeof(buf)
);
QString uri( (const char*)buf );

// create CORBA object
CORBA::Object_var obj = orb-
>string_to_object(uri.trimmed().toLatin1()
);
```

4.5 OMC API Enhancements

During the development of OMEdit several issues with the OMC Application Programming Interface (API) were discovered:

- Annotations for some models could not be retrieved via `getIconAnnotation`, `getDiagramAnnotation` or `getDocumentationAnnotation`.
- `addConnection` and `updateComponent` did not work correctly.
- `renameComponent` was very slow.
- The package `Modelica.UsersGuide` does not have any icon/diagram annotation but it has a non-standard Dymola annotation.

For example `getIconAnnotation(Modelica.Electrical.Analog.Resistor)` did not work because the `Resistor` model had component references inside the annotations. This problem was solved by symbolically elaborating (instantiating) the `Resistor` model, constant evaluating the `useHeatPort` parameter, and then elaborating the annotation record with this constant value.

Using constant evaluated parameters from elaborated model does not work for annotations that contain `DynamicSelect` and additional support for such annotations is needed. Unfortunately the `DynamicSelect` annotation creates problems for Modelica software that uses a client-server paradigm since it connects an annotation with a simulation, not with the actual model. However, `DynamicSelect` can still be handled by returning the entire expression to the client (here OMEdit) which could link a simulation variable to the annotation.

Retrieving the documentation annotation for MSL 3.1 did not work at first because these annotations hade been moved (MSL 2.x had no such requirements) to the end of the class definitions (typically in an equation section) and OMC only searched the public sections. This was solved easily in OMC by searching the entire model for the documentation annotation.

To make it easier to find which annotations cannot be retrieved correctly OMC was changed to return the exact annotation that was present in the model. Using this feature the problematic parts of the communication between OMEdit and OMC was debugged.

Updating components and adding connections to classes had small issues that were fixed to support OMEdit.

The package `Modelica.UsersGuide` and several others do not have any icon/diagram annotation. Displaying these packages in the MSL 3.1 browsing tree did not look nice. However, we observed that these packages has a non-standard Dymola specific annotation which is: `__Dymola_DocumentationClass = true`. In order to retrieve this annotation in OMEdit the OMC API had to be extended with a new function: `getNamedAnnotation(Modelica.UsersGuide) => true`. Now these packages can display a predefined icon in the tree browser.

To automatically test which component models have problems a script was written in OMEdit that walks the entire MSL 3.1 and calls OMC API functions on these models to see if the retrieved information was correct or not. A list with problematic models was built. Subsequently these issues were solved one-by-one.

The function to rename a component, `renameComponent` API function, was extremely slow when MSL 3.1 was loaded. This occurred because OMC had to go through all models and components and do a renaming refactoring. To resolve this and provide a faster functionality, we added a new API `renameComponentIn-`

`Class` that only renames the component locally in the model that is built using OMEdit and not in any other.

5 Annotations

Modelica annotations are used for storing auxiliary information about a model such as graphics, documentation or versioning etc. [2]. Once OMEdit is connected with OMC it can request the annotations. OMEdit uses three types of annotations;

- Annotations for Graphical Objects.
- Annotations for Connections.
- Annotations for Documentation.

5.1 Shapes/Component Models Annotations

All the shapes drawn in OMEdit are based on Modelica Annotations version 3.2. Graphical Annotations consist of two abstraction layers: the icon layer and the diagram layer. The icon layer contains the icon representation of a component and the diagram layer shows the inheritance hierarchy, connections, and inherited component models.

For example, a graphical icon representation of a `Ground` component model will look like this:

```
{-100.0,-
100.0,100.0,100.0,true,0.1,2.0,2.0,{Line(t
rue,{0.0,0.0},0,{{-
60,50},{60,50}},{0,0,255},LinePattern.Soli
d,0.25,{Arrow.None,Arrow.None},3,Smooth.No
ne),Line(true,{0.0,0.0},0,{{-
40,30},{40,30}},{0,0,255},LinePattern.Soli
d,0.25,{Arrow.None,Arrow.None},3,Smooth.No
ne),Line(true,{0.0,0.0},0,{{-
20,10},{20,10}},{0,0,255},LinePattern.Soli
d,0.25,{Arrow.None,Arrow.None},3,Smooth.No
ne),Line(true,{0.0,0.0},0,{{0,90},{0,50}},
{0,0,255},LinePattern.Solid,0.25,{Arrow.No
ne,Arrow.None},3,Smooth.None),Text(true,{0
.0,0.0},0,{0,0,255},{0,0,0},LinePattern.So
lid,FillPattern.None,0.25,{{-144,-
19},{156,-
59}},"%name",0,TextAlignment.Center)}}
```

This graphical representation of the `Ground` model is parsed by OMEdit for drawing this component model. The icon annotation is retrieved from OMC through the `getIconAnnotation` API command. Each graphical object is built up using the primitive graphical types; `Line`, `Polygon`, `Rectangle`, `Ellipse`, `Text` and `Bitmap` [2].

The primitive graphical types in OMEdit are handled through the `QGraphicsItem` class of Qt. A `ShapeAnnotation` class was created which is derived from `QGraphicsItem` and `QObject`. This class is an abstract class which contains classes of all primitive graphical elements.

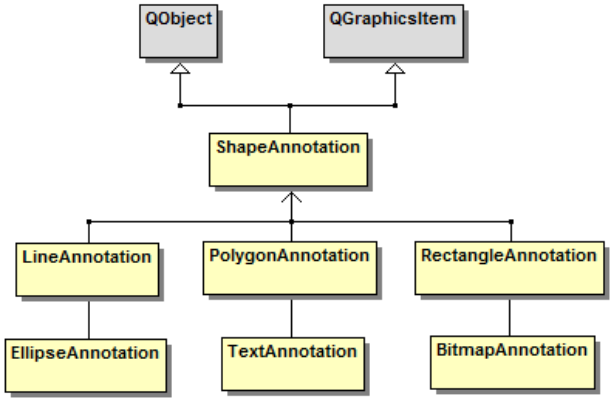


Figure 8. Classes hierarchy for predefined graphical elements.

5.2 Connection Annotation

The connection annotation defines the graphical representation of a connection between two component models. An example of connection annotation string is:

```
connect (a.x, b.x)
annotation(Line(points={{-25,30}, {10,30},
{10, -20}, {40,-20}}));
```

The connection annotation is composed of the primitive graphical type `Line`. The points of the line define the connection line co-ordinates between two connecting component models.

OMEdit creates an object of `Connector` class for each connection. Each `Connector` contains instances of `ConnectorLine` depending on the number of points in a connection. The `Connector` class is derived from `QGraphicsWidget` class which is container class for graphical objects. The `ConnectorLine` class is derived from `QGraphicsLineItem` which represents a single line. If we have `n` points in a connection annotation then we have `n-1` instances of `ConnectorLine`. In short `n` number of points creates `n-1` lines. The following shows the implementation of connection annotation in OMEdit.

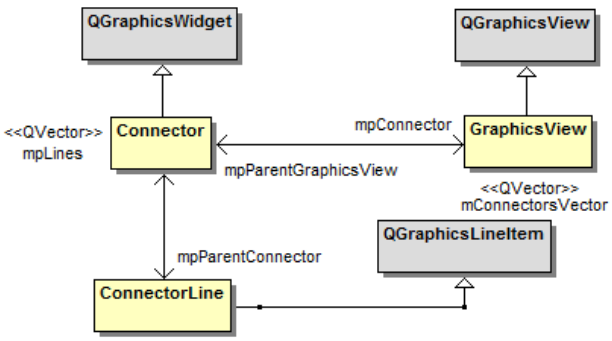


Figure 9. Implementation of connection annotation.

5.3 Documentation Annotation

The documentation annotation is used for textual descriptions of models. It is written as follows:

```
documentation_annotation:
annotation"(" Documentation "(" "info" "=" STRING
["," "revisions" "=" STRING ] ")" ")"
```

OMEdit requests OMC for the documentation of a specific component/library through the `getDocumentation` command and OMC returns the `info` annotation contained inside the documentation annotation which is a string. The tags `<HTML>` and `</HTML>` define the start and end of the string.

The `QWebView` class of Qt is used for displaying the HTML string of documentation annotation. The HTML string contains four types of links:

- Hyperlinks – Used to navigate to external websites.
- Image Links – Used to reference the local image files.
- Modelica Links – Used for linking to other component models.
- Mailto Links – Used to display email addresses that can be used for future contacts.

`QWebView` has built-in support for images so we didn't have to handle that. We just set the proper base path where all the images were located. However, for hyperlinks and mailto links we used the `QDesktopServices` class. This class uses the default system browser in case of hyperlink and default email client in case of mailto link. The Modelica links are special links which starts with `Modelica://` and reference to some component model or a package. Figure 10 shows the implementation of documentation annotation in OMEdit.

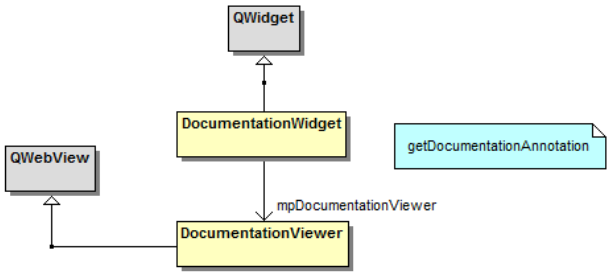
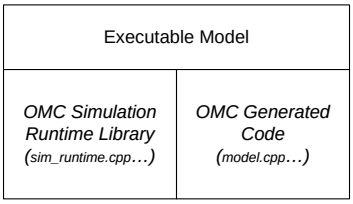


Figure 10. Implementation of documentation annotation.

6 Interactive Simulation

In order to offer a user-interactive and time synchronous simulation, OpenModelica has an additional subsystem to fulfill general requirements on such simulations, OpenModelica Interactive (OMI), shown in Figure 11. With OMI the user will be able to stimulate the system and interacting with it at runtime.

After creating and elaborating a Modelica model it is possible to simulate the model with OpenModelica. The outcome of calling the `simulate` or `buildModel` operation from the interactive session handler, is an executable, standalone C/C++ program generated from the internal simulation runtime code and the generated C/C++ model code by OMC (in this case `model.cpp`).



This executable contains the full Modelica model translated to C/C++ code based on all required equations, conditions and including different solvers. It offers both a non-interactive as well as an interactive simulation facility.

Since version 1.5.0 OpenModelica has an additional subsystem in order to offer a user-interactive and time synchronous simulation. This module is part of the simulation runtime core and is called "OpenModelica Interactive" (OMI). As mentioned above OMI will result in an executable simulation application, such as the non-interactive simulation. The following are some general functionalities of an interactive simulation runtime:

- The user will be able to stimulate the system during a running system simulation and to observe its reaction immediately.
- The simulation runtime behavior will be controllable and adaptable to offer an interaction with a user.
- A user will receive simulation results online during a simulation synchronous to real time, neglecting network process time and some other factors like scheduling of processes from the operation system.
- In order to offer a stable simulation, a runtime function will inform the user interface of errors and consequential simulation aborts.
- Simulation results will not under-run or exceed a tolerance compared to a thoroughly reliable value, for a correct simulation.
- Communication between a simulation runtime and a user interface will use a well defined interface and be based on common technology, in this case network communication.

In this case the `simulate` operation cannot be used. Instead the `buildModel` operation is needed.

To start an interactive simulation there is a need for more information, such as network configurations.

An important modification/addition to the semantics of the Modelica language during interactive simulation is the fact that parameters are changeable while simulating interactively using OMI. All properties using the prefix `parameter` can be changed during an interactive simulation. The fully qualified name is used as a unique identifier, so a parameter value can be found and changed regardless of its hierarchical position in the model. For more information see the OpenModelica System Documentation [1].

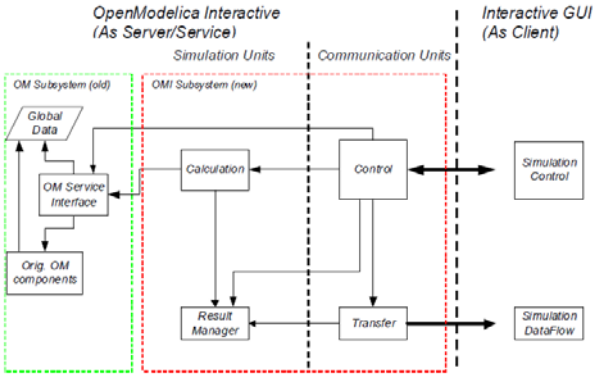


Figure 11. OpenModelica interactive system architecture overview.

7 Interaction with OMNotebook

OMEdit provides an environment where connection diagrams can be integrated with electronic interactive notebook. The idea is that the user performs the modeling in the connection editor and can subsequently export his/her models to an electronic notebook.

Alternately, the model in an electronic notebook is just an image. The model including its equations, algorithms, annotations etc. are hidden behind the picture. Thus, OMEdit is integrated with the OMNotebook tool [7], allowing users to click on the image and launch the model in connection editor where user can manage the connections, add/remove component models etc.

Figure 12 shows an electronic notebook with a `DCmotor` model as an image. When the user double clicks on the image, an OMEdit editing view is popped up, allowing both textual and graphical editing.

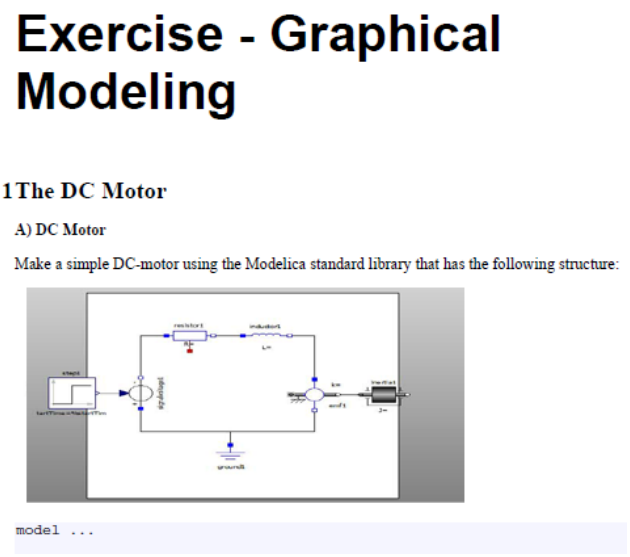


Figure 12. OMEdit integrated with OMNotebook used in a teaching material with exercises.

8 Related Work

There is previously one open source graphical editor available for OpenModelica:

- *SimForge* – Graphical and Textual Open Source Model Editor by Politecnico di Milano [3].

We have tried this editor for teaching, but found that the current implementation is too slow, not stable enough, and does not integrate with OMNotebook and interactive simulation.

There are also several commercial tools available for graphical modeling, e.g.:

- *Dymola* – Developed by Dynasim. Dymola, Dynamic Modeling Laboratory, is a complete tool for modeling and simulation of integrated and complex systems for use within automotive, aerospace, robotics, process and other applications [4].
- *MathModelica* – Developed by MathCore Engineering AB. MathModelica is a powerful, flexible and extensible system for multi-engineering modeling and simulation [5].
- *MapleSim* – High Performance Physical Modeling and Simulation from Maplesoft [6].

These are professional products that work well, but are not freely available, and are not open source. Also, they are typically not integrated with electronic books. An earlier version of MathModelica was integrated with the Mathematica electronic book, but did not provide interactive online simulation. The electronic notebook from Maplesoft is Maple-based and is pure textual. Also the Modelica language support is lacking in this tool.

9 Future Work

The first version of OMEdit is part of the OpenModelica 1.6 release. The version integrating OMEdit and Interactive simulation with OMNotebook will be available very soon, probably in the 1.6.1 release.

Moreover, somewhat improved 2D plotting is currently on the way. Future enhancements on the wish list include improved 3D graphic animation and support for displaying inheritance dependencies and sources of inherited equations and declarations.

10 Acknowledgements

This work has been supported by EU project Lila and Vinnova in the ITEA2 OPENPROD project. The Open Source Modelica Consortium supports the OpenModelica work.

References

[1] Adeel Asghar and Sonia Tariq. *Design and Implementation of a User Friendly OpenModelica Connection Editor*, master thesis LIU-IDA/LITH-EX-A-10/047-SE, Linköping University, Sweden, 2010.

[2] Open Source Modelica Consortium. *OpenModelica System Documentation Version 1.6*, November 2010. http://www.openmodelica.org

[3] Modelica Association. *The Modelica Language Specification Version 3.2*, March 24th 2010. http://www.modelica.org. Modelica Association. *Modelica Standard Library 3.1*. Aug. 2009. http://www.modelica.org.

[4] SimForge. http://trac.ws.dei.polimi.it/simforge/.

[5] Dymola. *Dynamic modeling tool*, http://www.dynasim.se.

[6] MathModelica. http://www.mathcore.com/products/mathmodelica/.

[7] Peter Fritzson, Johan Gunnarsson, Mats Jirstrand. MathModelica - An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming. In *Proceedings of the 2nd International Modelica Conference*, March 18-19, 2002, Munich, Germany.

[8] 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.