OPENMODELICA MDT ECLIPSE PLUGIN FOR MODELICA DEVELOPMENT, CODE BROWSING, AND SIMULATION

Elmir Jagudin, Andreas Remar, Adrian Pop, Peter Fritzson
PELAB – Programming Environment Lab, Dept. Computer Science
Linköping University, S-581 83 Linköping, Sweden
{adrpo, petfr}@ida.liu.se

Abstract: The OpenModelica Modelica Development Tooling (MDT) Eclipse plugin integrates the OpenModelica compiler with the Eclipse Integrated Development Environment Framework., giving additional capabilities for the model developer. MDT, together with the OpenModelica compiler, provides an environment for working with Modelica development projects. Simulation is possible from a special command window. To our knowledge MDT is the first Eclipse plugin for an equation-based language.

Keywords: OpenModelica, Modelica, modeling, Eclipse, simulation

1 INTRODUCTION

For a long time it has been an important goal to provide the model developer with better tool support to simplify or automate certain tasks.

The OpenModelica Modelica Development Tooling (MDT) Eclipse plugin is one step in this direction. It provides file and class hierarchy browsing and text editing capabilities. Some syntax highlighting facilities, code completion, and a compilation manager are also included in MDT. Recent enhancements to MDT, described in [8], are debugging and automatic indentation.

1.1 Eclipse Platform Architecture

The most important advantage of the Eclipse framework, see **Fig. 1** and [1], is that it is easy to add future extensions.

By itself, Eclipse does not provide a lot of end-user functionality. The important contribution of Eclipse is based on its plugins. The smallest architectural unit of the Eclipse platform is the plugin.

At the core of Eclipse is the Eclipse Platform Runtime. The Runtime in itself mostly provides the loading of external plugins. The Java Development Tooling (JDT) is for example a collection of plugins that are loaded into Eclipse when they are requested. The fact that Eclipse is in itself written in Java and comes with the Java Development Tooling as default often leads newcomers to believe that Eclipse is a Java IDE with plugin capabilities. It is in fact the other way around, with Eclipse being just a base for plugins, and the Java Development Tooling plugging into this base.

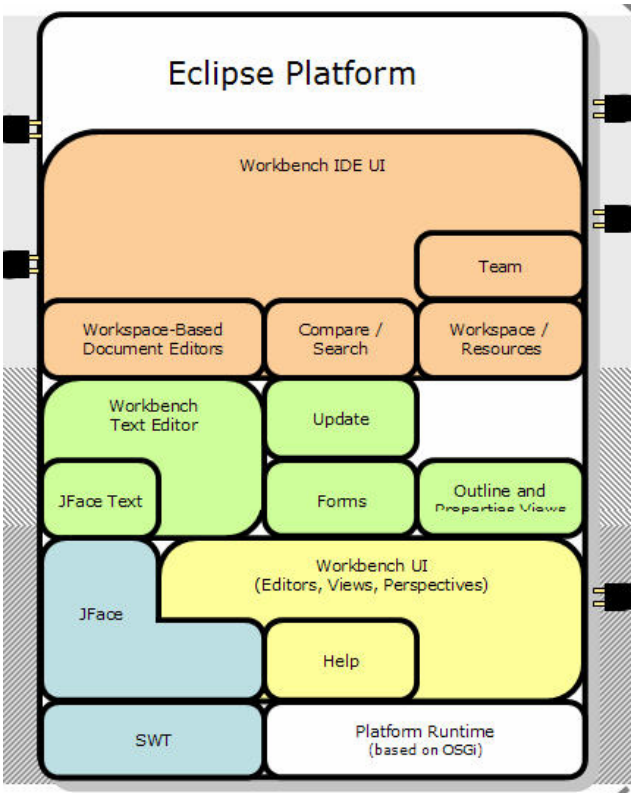

Fig. 1. The architecture of Eclipse, with possible plugin positions marked.

To extend Eclipse, a set of new plugins must be created. A plugin is created by extending a certain extension point in Eclipse. There are several predefined extension points in Eclipse, and plugins can provide their own extension points. This means that you can plug in plugins into other plugins.

An extension point can have several plugins attached, and what plugin that will be used is determined by a property file. For example, the Modelica Editor is loaded at the same time as the Java Editor is loaded. When a user opens a Java file, the Java Editor will be used, based on a property in the Java Editor extension. In this case, it is the file name extension that determines what editor that should be used.

As the number of plugins in Eclipse can be very large, a plugin is not actually loaded into memory before its contribution is directly requested by the user. This design make the memory impact reasonably low while running Eclipse.

A user-friendly aspect of Eclipse is the Eclipse Update Manager which allows you to install new plugins just by pointing Eclipse to a certain website. This website is provided by the developers of the plugin that you may wish to install. An update site at the OpenModelica web site is for example provided for easy installation of the latest version of MDT.

2 ECLIPSE HISTORY

In the mid 1990s software developments tools were primarily dominated by systems built around two technologies. Many of the tools were focused on a runtime environment developed and controlled by the Microsoft corporation. The other was built around the Java platform. The Java platform is less dominated by a single company and more open to industry and community input. IBM felt it was important to contribute to the growth of the more open Java platform to becoming dependent of Microsoft.

By creating a common platform for development tools built on top of the Java platform, IBM hoped to attract more developers from competing environments. In late 1998, the software division at the IBM corporation began working on the software project that is today known as Eclipse. The original work was based on resources developed by Object Technology International labs. In the beginning, work on a new Java IDE was done primarily at the IBM labs. In order to increase the rate of adaptation of the platform and to instill confidence in the Eclipse platform, IBM decided to release the code base under an open source license, and to build a community around the project.

In 2001, IBM together with eight other organizations created the Eclipse consortium. A website at `eclipse.org` was started in order to create and coordinate a community around Eclipse. The goal was that source code would be controlled and developed by the open source community and the consortium would handle the marketing and business side of the project.

At that point, IBM was the largest contributor to both the open source community and the consortium. Two years later the first major public release of the Eclipse platform was made. The release got a lot of attention from developers and was well received. However, industry analysts suggested that many were still perceiving Eclipse as an IBM-controlled technology. Many key players in the industry did not want to make commitments to a project controlled by the International Business Machines corporation.

After discussions within the consortium it was decided that a new organization was needed to make the status of Eclipse as an open and community driven project clear. At the EclipseCon 2004 gathering an announcement was made that the Eclipse Foundation was formed. The foundation is an independent not-for-profit organization. It has its own full time paid professional staff, supported by foundation members.

The new organization has proven itself a success. At this point the foundation has released version 3.0 and 3.1 of Eclipse since its birth. These releases have gained more adaptation and recognition than any earlier versions. Today (2005) the foundation has more than 90 full-time developers on the pay roll and receives more than $2 millions in funding each year.

Currently there are more than eighty member companies in the foundation of which at least sixty-nine are providing add-on products to Eclipse. Today there exists hundreds of proprietary and an even greater number of free plugin products. Eclipse has gained a strong foothold in the industry and is one of the major open source software development platforms [2].

3 OPENMODELICA ENVIRONMENT ARCHITECTURE

MDT is integrated in the OpenModelica environment which consists of several interconnected subsystems, as depicted in Fig. 2.

Arrows denote data and control flow. Several subsystems provide different forms of browsing and textual editing of Modelica code.

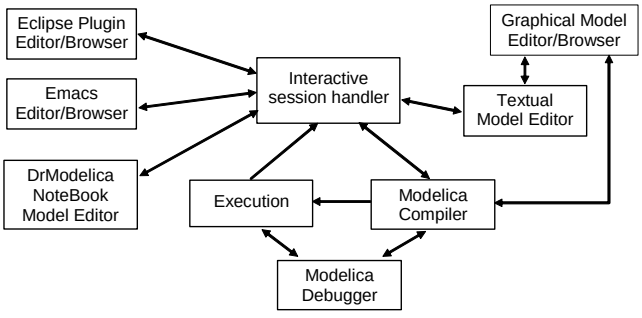

Fig. 2. Architecture of OpenModelica environment.

4 MODELICA DEVELOPMENT TOOLING (MDT) ECLIPSE PLUGIN

As mentioned, the Modelica Development Tooling (MDT) Eclipse plugin provides an environment for working with Modelica development projects.

The following MDT 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 when writing code to reference a class or function
- Browsing of the Modelica Standard Library

Moreover, debugging and indentation support has recently been added [8].

4.1 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`.

4.2 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`.

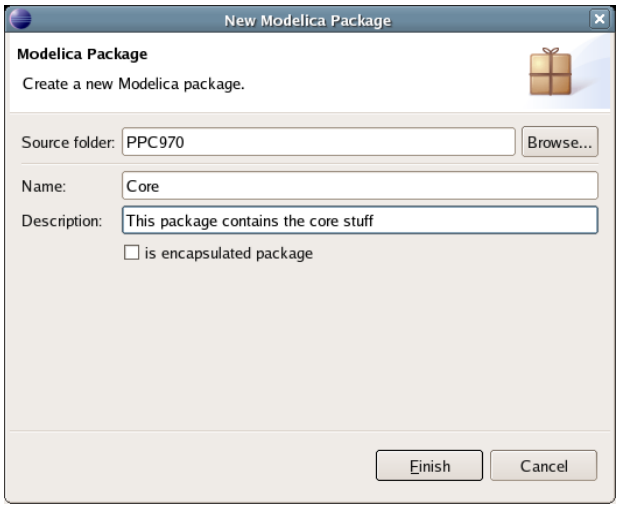

Fig. 3. Creating a new package.

4.3 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.

4.4 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`.

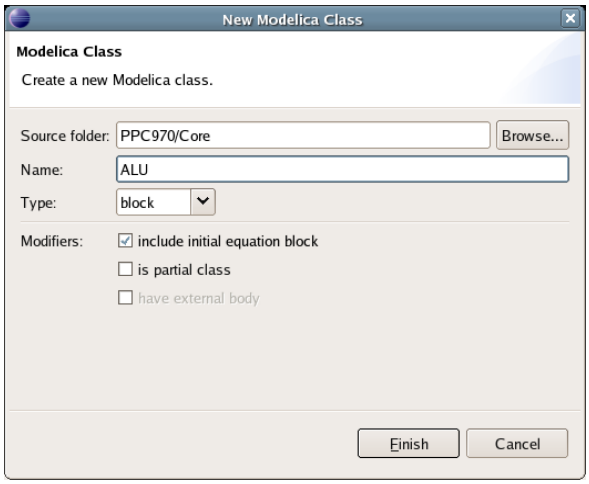

Fig. 4. Creating a new class.

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.

4.5 Syntax Checking

Whenever a Modelica (`.mo`) file is saved by the Modelica Editor, it is checked for syntactic 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.

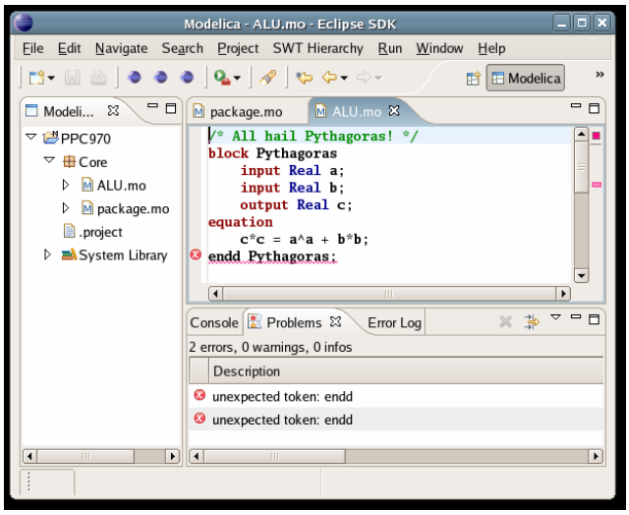

Fig. 5. Syntax checking.

4.6 *Code Completion*

MDT supports Code Completion in two variants. The first variant, code completion when typing a dot after a class (package) name, shows alternatives in a menu:

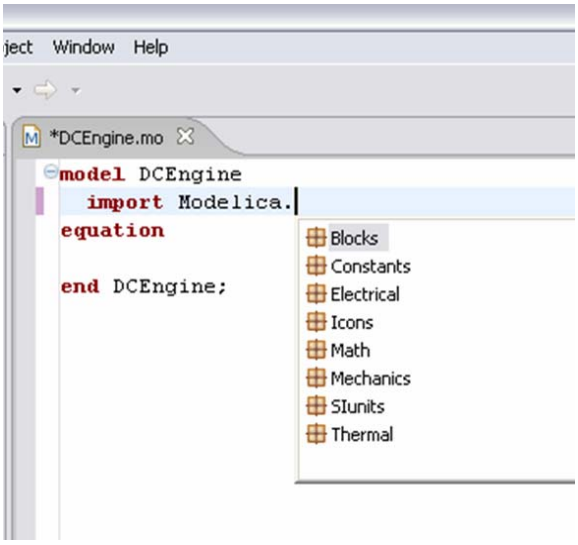

Fig. 6. Code completion using a popup menu after a dot.

The second variant is useful when typing a call to a function. It shows the function signature (formal parameter names and types) in a popup when typing the parenthesis after the function name, here the signature `Real sin(SI.Angle u)` of the `sin` function:

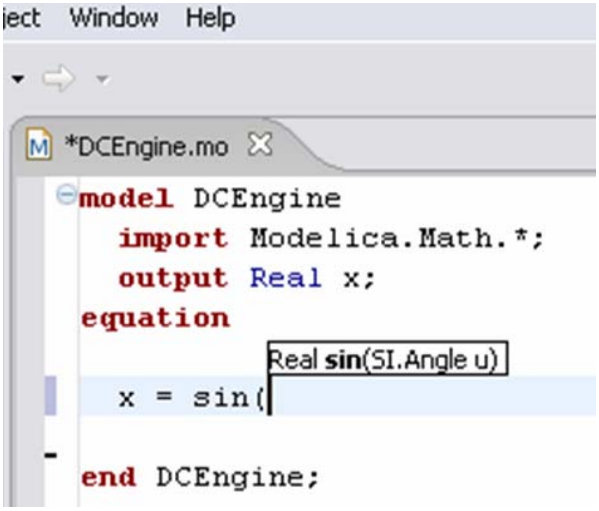

Fig. 7. Code completion showing a popup function signature after typing a left parenthesis.

5 SIMULATION AND PLOTTING FROM MDT

Simulation and plotting is possible from a special command window, where commands are sent to the OpenModelica Compiler omc. For example:

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. 8**.

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

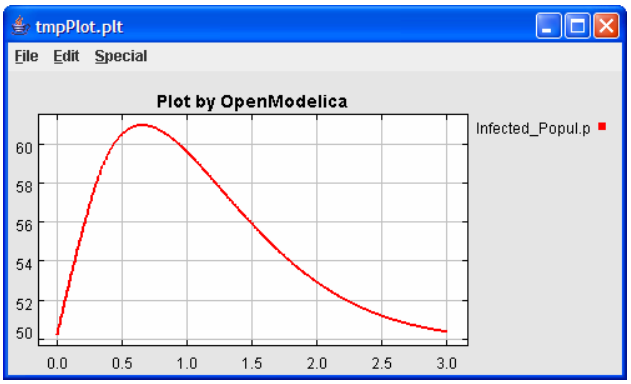

Fig. 8. Plot of simulation from the Influenza model.

6 DISCUSSION

As mentioned, the main idea behind the development of MDT is to make the benefits of software development support in Eclipse also available to model developers, especially developers of large models. A natural

question is whether these promises been fulfilled. It is still too early to give definite answers to this question since the MDT became available very recently, a few months ago, and some of the more advanced facilities such as code completion are just now becoming available. So far, we have positive experience of early users working in the MetaModelica [9] extended subset of Modelica on large (100 000 line) applications, as well as using standard Modelica on small application models, which is very promising.

7 CONCLUSIONS

An integrated development environment for Modelica as a plugin to Eclipse has been developed. It is aimed for development of large models, is called MDT (Modelica Development Tooling), and is part of OpenModelica. MDT allows browsing, editing, building executables, and simulation from a special command window. It also has facilities such as code completion, query, and automatic indentation.

ACKNOWLEDGEMENTS

This work was supported by Vinnova in the SWEBProd project, by SSF in the VISIMOD and ProViking SECD projects, by the CUGS graduate school, and by MathCore Engineering AB.

REFERENCES

[1] Eclipse website. http://www.eclipse.org.

[2] Eclipse history. A brief history of eclipse. http://www128.ibm.com/developerworks/rational/library/nov05/cernosek/

[3] Peter Fritzson, Peter Aronsson, Håkan Lundvall, Kaj Nyström, Adrian Pop, Levon Saldamli, David Broman. The OpenModelica Modeling, Simulation, and Development Environment. In *Proceedings of the 46th Conference on Simulation and Modeling of the Scandinavian Simulation Society* (SIMS2005), Trondheim, Norway, October 13-14, 2005. www.ida.liu.se/projects/OpenModelica

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

[5] Peter Fritzson, *et al.* The OpenModelica Users Guide, version 0.7, May 2006. www.ida.liu.se/projects/OpenModelica

[6] Peter Fritzson, *et al.* The OpenModelica System Documentation, version 0.7, May 2006. www.ida.liu.se/projects/OpenModelica

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

[8] 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.

[9] Adrian Pop and Peter Fritzson. MetaModelica: A Unified Equation-Based Semantical and Mathematical Modeling Language. To appear in *Proceedings of Joint Modular Languages Conference 2006* (JMLC2006) LNCS Springer Verlag. Jesus College, Oxford, England, Sept 13-15, 2006.