Brück D., Elmqvist H., Olsson H., Mattsson S.E.:

Dymola for Multi-Engineering Modeling and Simulation

2nd International Modelica Conference, Proceedings, pp. 55-1 – 55-8

Dymola for Multi-Engineering Modeling and Simulation

Dag Brück, Hilding Elmqvist, Sven Erik Mattsson and Hans Olsson
Dynasim AB, Research Park Ideon, SE-223 70 Lund, Sweden
E-mail: info@dynasim.se

Abstract

Dymola is an integrated environment for developing models in the Modelica language. The growing use of Dymola has over time increased the demands on the development environment. Requests for extension and redesign originate from two sources: the need to simplify the use of Dymola to better support new and inexperienced users, and the need to better support “power users” which model extremely large and complex systems.

Key areas in the development of Dymola are: a simplified and more coherent graphical user interface, browsing facilities for navigating large and complex systems, new experiment facilities for managing complex simulation tasks, distributed (parallel) simulation, and integrated version control to help manage model libraries and complete models.

The paper describes the extensively redesigned Dymola 5, with an emphasis on new features compared to Dymola 4.

Introduction

Dymola is an integrated environment for developing models in the Modelica language [Modelica Association, 2002; Tiller, 2001], and a simulation environment for performing experiments. It is used since several years within major companies for complex simulations. For example, Dymola has been used to simulate detailed models of complete vehicles including engine, transmission and chassis [Tiller et al., 2000].

Dymola uses hierarchical object-oriented modeling to describe, in increasing detail, the systems, subsystems and components of a model. Reuse of modeling knowledge is a key issue, and is supported by use of libraries containing model classes and by the use of inheritance. Physical couplings are modeled by defining physical connectors and graphically connecting submodels.

Model libraries are available for electronics, rotational, translational and 3D mechanics, thermodynamics, hydraulics and control systems. The libraries range from basic components to more specialized domains such as the power train library. Predefined libraries can be expanded with user-written model libraries.

The growing use of Dymola has over time increased the demands on the development environment. Requests for extension and redesign originate from two sources:

- The need to simplify the use of Dymola to better support new users and inexperienced users. This is of particular importance when Dymola is used for teaching.

- The need to better support “power users” which model extremely large and complex systems. In this case, the user needs significant support from the environment to handle very large amounts of information, to document complex systems, and to verify results. The development of large component libraries is a collaborative effort involving several people, which requires adequate tool support. Also, different software packages are used which underlines the need for information exchange.

Key areas in the development of Dymola are:

- Simplified graphical user interface. In addition to better structuring, the use of modern GUI elements (help facilities, dockable windows etc.) makes it easier to use the program.

- Browsing facilities for navigating large and complex systems. This includes class browsers for navigating component libraries and a new model browser for navigating complex models.

- New experiment facilities for managing complex simulation tasks. They handle multiple parameter sets, models of different complexity, and tools for validating models.

- Distributed simulation on several computers, allowing parallel simulation for tasks such as optimization.

- Integrated version control to help manage model libraries and complete models. The user needs support for version control to store/retrieve models and associated data, to compare versions of a model, plus mechanisms for documenting the evolution of models.

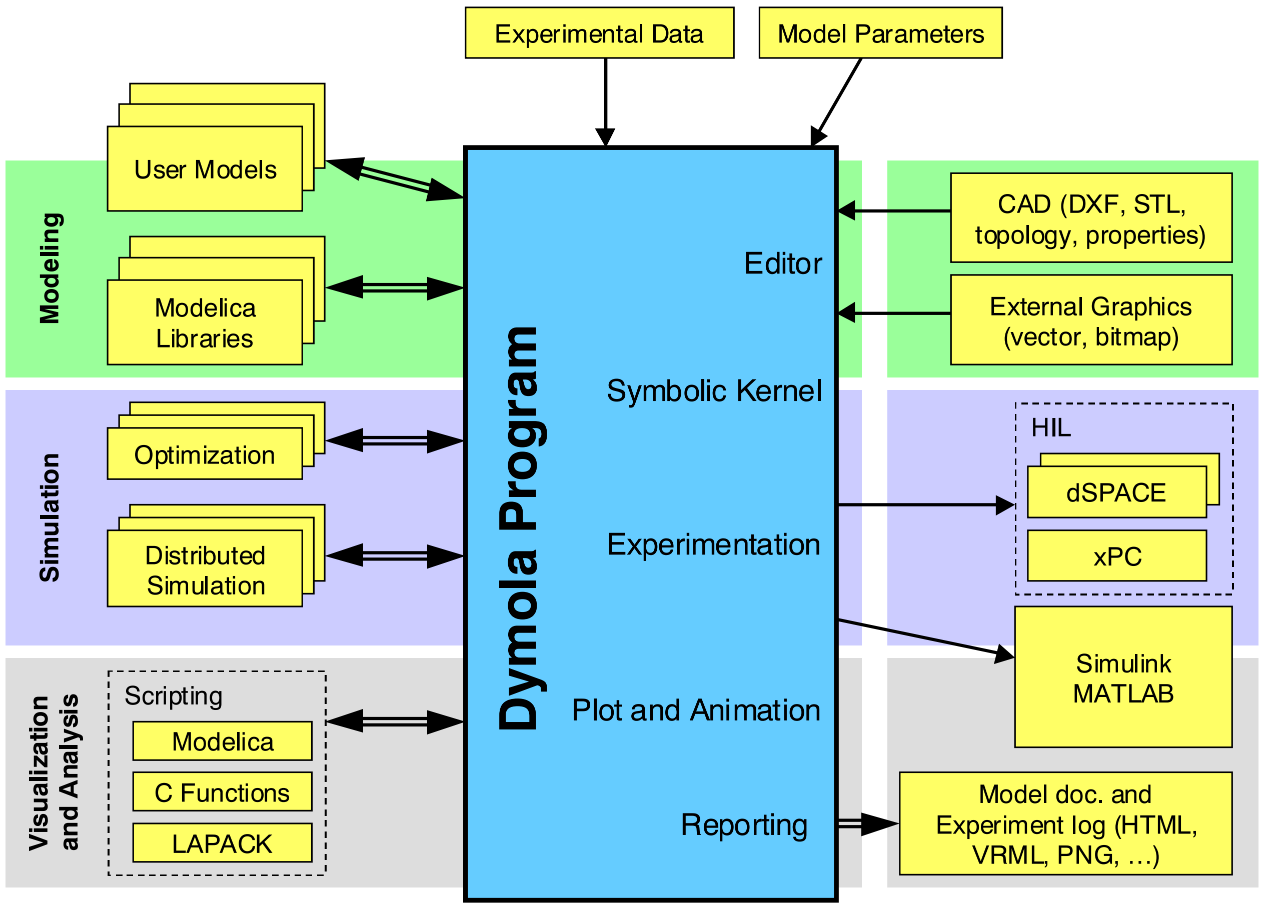


Figure 1. The Dymola architecture.

Dymola architecture

Dymola is an integrated environment for modeling and simulation. Figure 1 describes the architecture and connectivity of Dymola 5.

At the **modeling level**, models are composed from library components (from the Modelica standard library, other free libraries, commercial and proprietary libraries), as well as models developed by the user. Models are either composed of other, more primitive, components, or described by equations at the lowest level. The equation-based nature of Modelica is essential for enabling truly reusable libraries. Measurement data and model parameters cover additional model aspects.

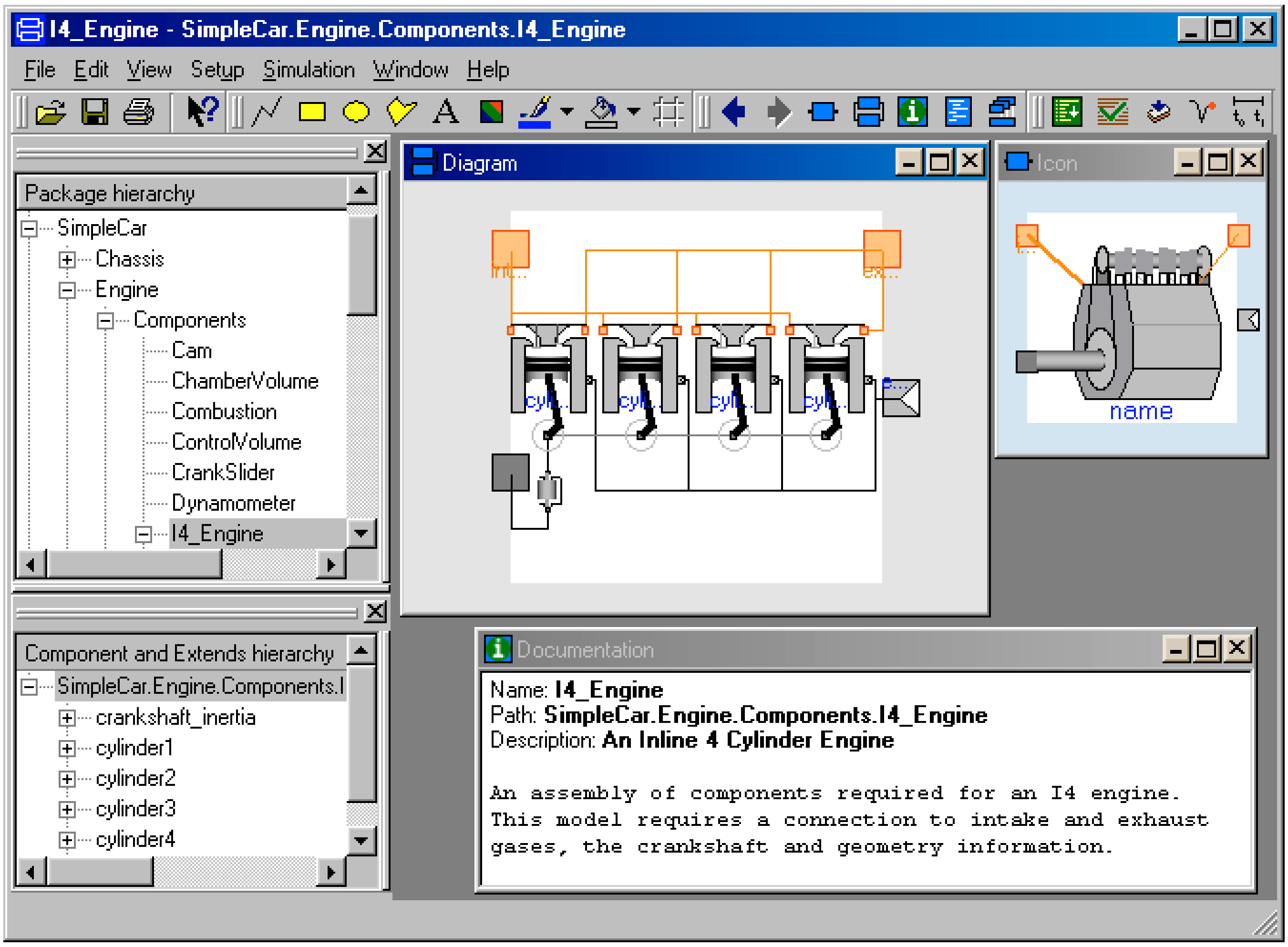
Detailed model knowledge can be imported from CAD packages. Examples of such information are mass and inertia of 3D mechanical bodies, and the topology of a multibody system (bodies and joints). Graphical properties may be described in DXF and STL format. The icons of model components are defined either by drawing shapes in Dymola, or by importing graphics from other tools in common vector or bitmap formats.

At the **simulation level**, Dymola transforms a declarative, equation-based, model description into efficient simulation code. Advanced symbolic manipulation (computer algebra) is used to handle very large sets of equations. Efficient simulation, including realtime simulation of hydraulic systems, can only be achieved after extensive symbolic transformations of the equations [Elmqvist et al., 2002].

Dymola provides a complete simulation environment, but can also export code for simulation in Simulink. In addition to the usual offline simulation, Dymola can generate code for specialized Hardware-in-the-Loop (HIL) systems, such as, dSPACE, xPC and others.

Recent developments in Dymola 5 allow distributed (parallel) simulation on several computers in a network, for example to perform parameter studies. There are facilities for optimization, also carried out with parallel simulation runs. Such experiments are controlled with a Modelica-based scripting language, which combines the expressive power of Modelica with access to external C libraries, e.g., LAPACK.

The built-in plotting and animation features of Dymola provide the basis for **visualization and analysis** of simulation data. Experiments are documented with logs of all operations in HTML format, including animations in VRML (Virtual Reality Modeling Language) and images. Models and libraries are extensively documented in HTML automatically generated by Dymola from the models themselves.



Name: **I4_Engine**
Path: **SimpleCar.Engine.Components.I4_Engine**
Description: **An Inline 4 Cylinder Engine**

```
An assembly of components required for an I4 engine.
This model requires a connection to intake and exhaust
gases, the crankshaft and geometry information.
```

Figure 2. The model editor.

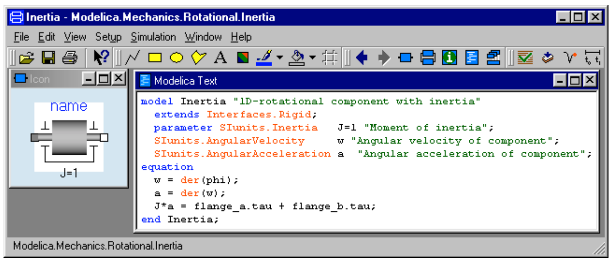
Graphical user interface

The graphical user interface has been extensively redesigned. In Dymola 5 emphasis has been put both on simplifying the task of building models for the novice user and on providing tools for building and managing large and complex models developed by a collaborating team of engineers.

Graphical editor

Figure 2 shows a screen dump of the Dymola modeling environment. The top left tree browser shows the (Package) hierarchy of a library called SimpleCar [Tiller, 2001]. When I4_Engine is chosen different representations (icon and composition diagram) of the model I4_Engine are shown. The lower left tree browser, "Component and Extends hierarchy", shows the hierarchical decomposition, for example, that the engine model contains crankshaft-inertia and the four cylinders: cylinder1, … cylinder4. A visual representation of that is shown in the Diagram in the middle. An Icon representation of the engine is shown at the top right. A The Documentation window is shown at the lower right. Such a documentation window contains HTML formatted information, i.e. also graphics and links to other resources may be included.

Editing of models at the fundamental level has been improved by syntax highlighting of the Modelica code, see Figure 3. Another convenience is that models can be dragged from the package browser into the text editor, which gives access to fundamental types in the Modelica library with no typing. Editing in the textual view is instantaneously represented in the graphical view.


Inertia - Modelica.Mechanics.Rotational.Inertia
File Edit View Setup Simulation Window Help
Icon
name
J=1
Modelica Text
Modelica.Mechanics.Rotational.Inertia

```
model Inertia "1D-rotational component with inertia"
  extends Interfaces.Rigid;
  parameter SIunits.Inertia   J=1 "Moment of inertia";
  SIunits.AngularVelocity     w "Angular velocity of component";
  SIunits.AngularAcceleration a  "Angular acceleration of component";
equation
  w = der(phi);
  a = der(w);
  J*a = flange_a.tau + flange_b.tau;
end Inertia;
```

Figure 3. Model editor with syntax highlighting.

The Icon representation can be created with a built-in graphical editor. It allows insertion of lines, rectangles, ellipses, polygons and text strings. Figure 4 shows the tool bar for the graphical editor.

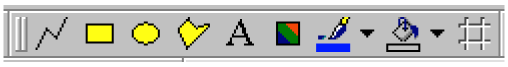


Figure 4. Drawing tools.

It is also possible to insert scalable bitmaps created in other tools like MS Paint and scalable vector graphics from the clipboard. Advanced graphics can thus be created in, for example, MS PowerPoint or MS Visio and inserted into Dymola as Icons or backgrounds for the composition diagrams.

The toolbar also contains controls for setting graphical attributes, e.g., foreground and background color, line style and fill pattern.

As indicated above, Dymola 5 supports Modelica's notion of different layers of information:

- Icon layer
- Diagram layer
- Documentation layer
- Modelica text layer
- Model dependencies layer (generated by Dymola)

It should be noted that Dymola 5 allows several layers to be shown simultaneously.

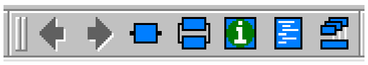


Figure 5. Navigation tools.

Figure 5 shows the buttons of the navigation tool in Dymola. The first two buttons are used to navigate in the component hierarchy, similar to navigation with a web browser. The back arrow displays the previously visited component; the forward arrow negates the backward move. The other buttons are used to display layers in the graphical editor

Simplifications

In response to user comments, a major design goal was to simplify the graphical user interface. The first step has been to reduce the number windows: both model editing and simulation is controlled from a single window, and plot/animation windows are not opened until a simulation has been performed (or opened explicitly by the user). The design has been influenced by common paradigms, for example, the web-browser approach to navigation.

The design of Dymola 5 more closely follows published guidelines [Microsoft, 1999], and has in general adopted more modern idioms compared to Dymola 4. Common operations are invoked by buttons in addition to menu commands. Dockable windows which either can be part of the main editor window, float on the desktop or be minimized, are used for browsers and similar tools.

The extended use of commonly used GUI elements (toolbars, dockable windows, "what's this" help information) makes Dymola consistent with other applications.

Browsing

The "Package hierarchy" browser shows the library structure and it is possible to drag a component model from the tree into a Diagram in order to add a component to a model, see Figure 6. The browser can either be docked to the editor window as shown in Figure 2, or be dragged onto the desktop.

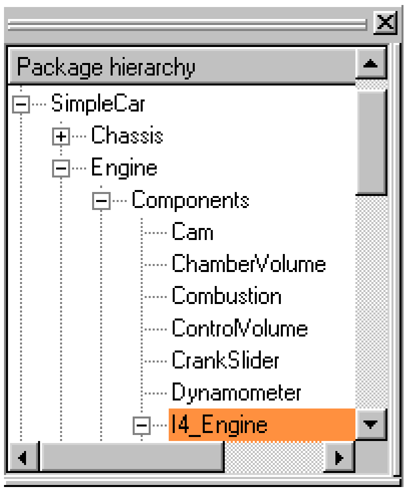


Figure 6. The package browser.

The components of a library can also be viewed as icons in a separate library window, see Figure 7, from which components can be dragged.

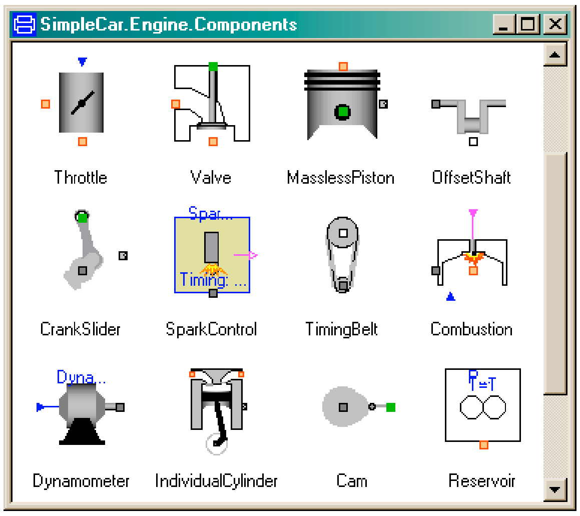
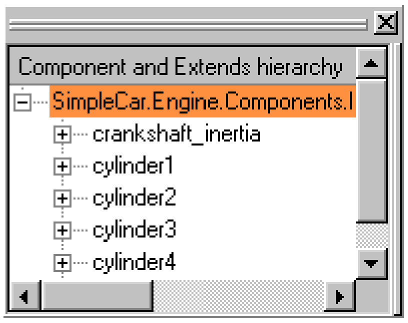


Figure 7. Library window.

The hierarchical structure of a model is shown in the "Component and Extends hierarchy" browser. The top-level components of an engine model are shown in Figure 8.

Maneuvering in this hierarchical structure can be done by clicking in the tree which then changes the view to the selected model. It is also possible to point at an icon and "zoom-in" on the content, i.e. next abstraction layer.

When a model is chosen in the package browser, it becomes the root model of the graphical editor. The root model is used in check, translate and simulate



Component and Extends hierarchy
SimpleCar.Engine.Components.I
crankshaft_inertia
cylinder1
cylinder2
cylinder3
cylinder4

Figure 8. The component browser.

commands. Navigation into its component hierarchy allows inspection of model details, but does not change the root model or permit editing. This view is consistent with the common metaphor used in web browsers.

Dymola 5 has search facilities, for example to search for models that mention particular keywords in the documentation. It may also be useful to find models with a component or a parameter with a known name.

For advanced users, the biggest problem has been to organize the large amount of information in complex models and extensive component libraries. The biggest improvement in Dymola 5 is the use of hierarchical browsers for navigating packages and models. The package browser is also the natural focal point for copying/renaming of models and restructuring of packages.

Advanced Modelica concepts, such as, replaceable classes, is given an intuitive user interface via the component browser. If a class is declared as replaceable, the actual class can be dragged from the package browser onto the replaceable class in the component browser. Other features that benefit from the new user interface are choices (a selection of replaceable classes) and arrays of components.

Visualization in 3D

The graphical editor represents a abstraction of the model, the object diagram. When building 3D mechanical systems, the user greatly benefits from the instantaneous 3D visualization available in Dymola 5. Parameters settings for e.g. the length of a bar can be visually checked in the animation window.

Experimentation

By "experimentation" we mean all the steps necessary to use a model in order to achieve useful results. That includes setting up model parameters and initial conditions, running simulations, analysis of simulation data, and report generation.

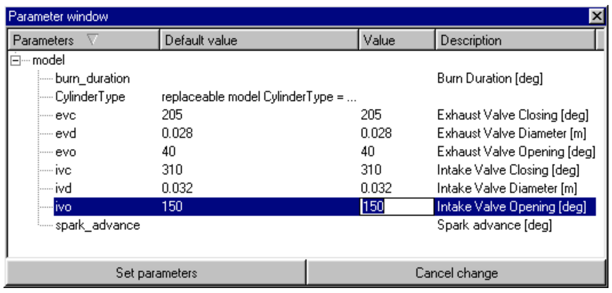
Parameter values specific to the studied model have to be entered in a form associated with a component, see Figure 9. Parameters and initial conditions can be set at three different abstraction levels:

• The default values specified in the model of a component, when a reasonable default exists.

• Parameter values that are specified in the modifier list of a specific component. For example, the crankshaft shift is different for each cylinder in an engine.

• Model parameters which are specific for a given top-level model. Such parameters are specified at the top-level of the model, and then propagated through a hierarchical modifier.

Dymola allows the user to set parameters and initial conditions at each of these levels, either through the model editor or while running simulations.



Parameter window

Parameters	Default value	Value	Description
model			
burn_duration			Burn Duration [deg]
CylinderType	replaceable model CylinderType = ...		
evc	205	205	Exhaust Valve Closing [deg]
evd	0.028	0.028	Exhaust Valve Diameter [m]
evo	40	40	Exhaust Valve Opening [deg]
ivc	310	310	Intake Valve Closing [deg]
ivd	0.032	0.032	Intake Valve Diameter [m]
ivo	150	150	Intake Valve Opening [deg]
spark_advance			Spark advance [deg]

Figure 9. Parameters for specification of details of the engine

For visualization, Dymola offers plotting and 3D animation. Figure 10 shows a window with multiple

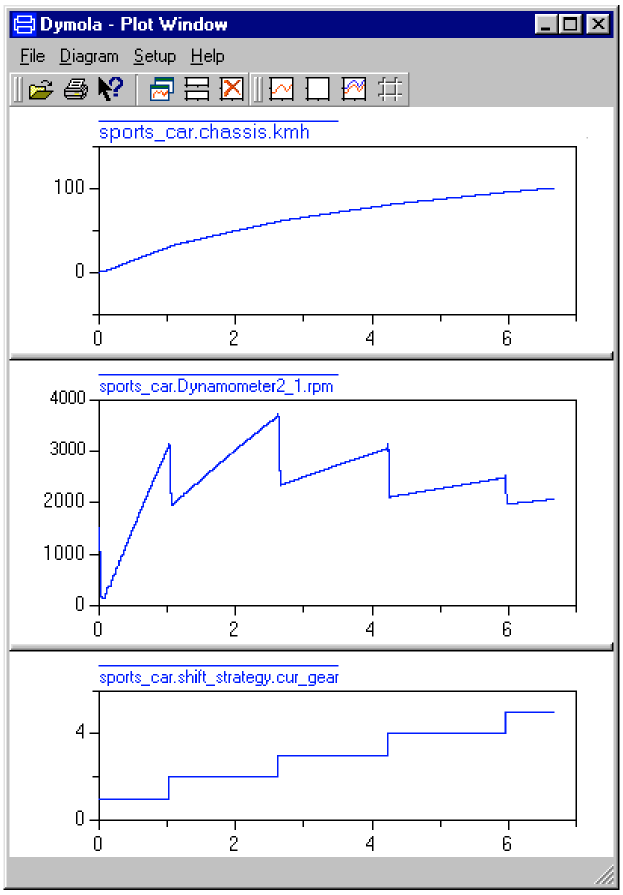


Figure 10. Plot of car speed, engine RPM and selected gear versus time.

plots of car speed, engine RPM and selected gear versus time during such an experiment. The car accelerated to 100 km/h in 6.66 seconds. Plots can be exported as PNG files for inclusion in session log or as vector graphics.

Animation is provided by specialized visualization properties which are present in the mechanical libraries by default. These properties are calculated during simulation and then used to show 3D views in Dymola, as shown in Figure 11. It is also possible to export such animations in VRML format [VRML, 1997], which can be examined with special viewers or with plugins for web browsers.

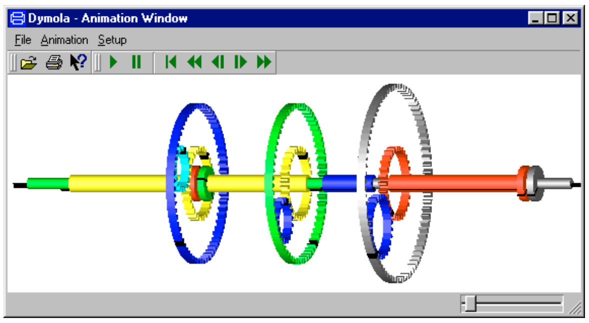
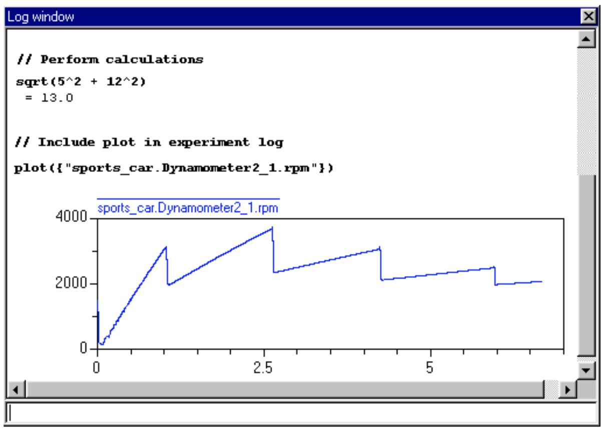


Figure 11. Animation of an automatic gearbox.

Dymola 5 has powerful features for postprocessing of simulation results. It is possible to compare simulation results with experimental data. Data can be imported and exported to other programs like Matlab and Microsoft Excel. There is a scripting language based on Modelica for automating design studies and analysis. Interfaces to subroutine packages such as LAPACK (or other libraries written in C or FORTRAN) enables advanced numerical calculations. The scripting language is also used for running parameter studies in a distributed environment (see below) and for performing optimization.


Log window

```
// Perform calculations
sqrt(5^2 + 12^2)
 = 13.0

// Include plot in experiment log
plot({"sports_car.Dynamometer2_1.rpm"})
```

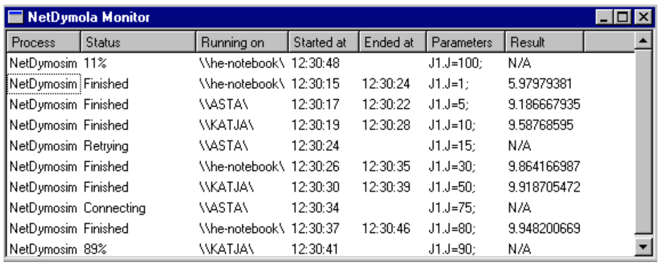
Figure 12. Dymola session window

Automatic logging of design sessions including graphics is provided as HTML code for archiving and sharing over the Internet, see Figure 12. A complete experiment report can be written by editing the session log.

Distributed simulation

During the design phase, hundreds or thousands of simulations have to be performed with different parameter sets. Optimization is used to determine parameters in the model by fitting simulation results to experimental data and to optimize the parameters of a design. It is a task that significantly benefits from parallel simulation. Dymola 5 can use many computers and automatically schedule simulations in parallel to shorten the design cycle.

Figure 13 shows the Dymola monitoring window for parallel simulations. It shows the status of each simulation run: the parameters used and optional criteria result. The Dymola scheduler assigns tasks to computers as they become available. When a simulation finishes, the next task is run on the freed computer. Transfers of the simulation code, input data (parameters and initial conditions) and results are fully automated.


NetDymola Monitor

Process	Status	Running on	Started at	Ended at	Parameters	Result
NetDymosim	11%	\\he-notebook\	12:30:48		J1.J=100;	N/A
NetDymosim	Finished	\\he-notebook\	12:30:15	12:30:24	J1.J=1;	5.97979381
NetDymosim	Finished	\\ASTA\	12:30:17	12:30:22	J1.J=5;	9.186667935
NetDymosim	Finished	\\KATJA\	12:30:19	12:30:28	J1.J=10;	9.58768595
NetDymosim	Retrying	\\ASTA\	12:30:24		J1.J=15;	N/A
NetDymosim	Finished	\\he-notebook\	12:30:26	12:30:35	J1.J=30;	9.864166987
NetDymosim	Finished	\\KATJA\	12:30:30	12:30:39	J1.J=50;	9.918705472
NetDymosim	Connecting	\\ASTA\	12:30:34		J1.J=75;	N/A
NetDymosim	Finished	\\he-notebook\	12:30:37	12:30:46	J1.J=80;	9.948200669
NetDymosim	89%	\\KATJA\	12:30:41		J1.J=90;	N/A

Figure 13. Dymola monitoring window for parallel simulations.

During normal simulation on a single computer, a simulation is performed through cooperation between the Dymola program and a separate simulation process. In a distributed environment, a third party, known as the simulation proxy, handles data transfers between Dymola and the simulation task; the use of a proxy allows exactly the same simulation code to run locally and on another computer. As a special case, the "distributed" scheme can utilize multiple CPUs on one computer.

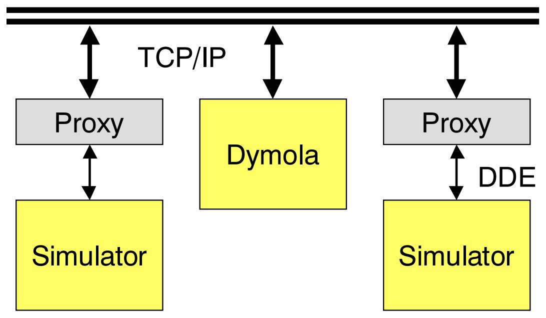


Figure 14. Architecture of distributed simulation.

A proxy is started on each machine willing to act as "compute server", see Figure 14. On receiving a connection via TCP/IP from a Dymola program, its first task is to help copy the simulation code and input files to a unique area on the server. It then relays parameter settings and commands from Dymola, and handles data transfers from the simulation to Dymola for online animation and plotting. The Dymola program maintains a list of computers that may be asked to run simulations; the user can control this list by simple commands.

This scheme for distributed simulation is designed for cooperative sharing of resources and quite simple; security measures are limited. First, a computer can only be used as server after the proxy has been started. Second, the proxy runs as an unprivileged process, having only the capabilities of the user starting it. Load is limited because each proxy blocks requests while a simulation is running, but it is possible to start additional proxies to handle multiple simulations (e.g., if the computer has multiple CPUs). Ways to utilize existing system security features need to be further investigated.

Collaborative development

In developing model components for a complex system such as a vehicle, many different kinds of competence are needed. Experts in engines, transmissions and chassis etc. are needed. Because several people are involved in the process, it becomes essential to break up or decompose the overall problem into modular units during development.

The equation-based modeling supported by the Modelica language is fundamental in enabling true reuse of modeling knowledge and the practical use of model libraries. Dymola is able to transform equations of subcomponents as required by the structure of the system. Without the equation-based foundation, several variants of a single model are needed to handle different computational causality. Even worse would have been that the user of a library is given the responsibility to analyze the computational causality of the system in order to pick the right variant.

Inheritance is also important for supporting reuse. Model libraries may include partial models that describe common properties of a set of component types. Such a partial model is conveniently used as a base class to develop models for the individual types of the set by just adding a specific part that distinguish it from the others in the set. This approach makes it simpler to add new component models as well as simplifies maintenance since the common properties of the component types are described only once.

Furthermore, as more people are involved in the process, the development is geographically and chronologically distributed because it is natural to have centers with specific core-competencies. This implies that the modular units developed must be seamlessly integrated to solve the overall problem, and the partitioning should be able to reflect the organizational structure of the model development teams.

In order to increase quality and reduce development time, tools should be made available to

- Provide a structure for organizing, storing and retrieving information (models, documentation, experiment data).
- Support the exchange of information and simplify reuse of models throughout the organization.
- Ensure that correct information is available to each user (versions of libraries, corresponding experiments).

A version control system provides means to track changes to a set of files. A "commit" operation associates a developer and documentation with each change to the common storage of files. The Modelica text of two versions can be compared, and it is possible to back up to any previous version.

The underlying version control system must be able to support multiple concurrent developers working on the same set of models. Extensive locking of files is undesirable in a collaborative environment, and more recent tools also support concurrent development of closely related parts (with appropriate safety nets). A single physical person may have multiple roles in the development or use of the library.

Tracability is essential for maintaining quality over time. Tool enforcement to document modifications before they become publicly available gives the opportunity to review changes and improves quality. The development history and documentation of changes may also be needed for tracing model incompatibilities, for example.

Model testing should be integrated with model development, which implies that the version control system must be able to handle test scripts, support utilities and binary test data. Regression testing, where models are simulated and compared with known good simulation results, is very powerful in detecting involuntary changes to model libraries. A failed regression test may cause either a change of a model, or the revision of the test itself.

Multiple libraries are often used together. In this case, version compatibility across libraries becomes essential. It must be possible to "tag" releases of multiple libraries to indicate compatibility at the project level.

Dymola will support storing, retrieving, etc. of models in version control systems such as CVS (Concurrent Version System) [CVS]. We have deliberately chosen to build on existing version control systems, which offers greater flexibility and better integration than a proprietary system. Because of the textual representation of models in the Modelica language, existing text-based tools can be used, for example, to compare versions. To browse changes in large systems, support in the graphical environment of Dymola is needed.

The use of public libraries has increased in industry over several years. More recent is "open source development", which can be described as the loosely organized development (typically of software) by several geographically separated parties. Public websites, such as SourceForge, support Open Source development with web-based tools and CVS. The Modelica Standard Library is maintained as a project at SourceForge.

Library protection

There are many closed simulation packages on the market where you are not able to see what model is used. Modeling is an art in the sense of describing the relevant aspects of the object under observation. It is thus very important to be able to see what assumptions and approximation that the author of a model made. Dymola is open to view all and possibly modify the details by showing of the Modelica code. However, if a company want to protect proprietary information when shipping models, Dymola will support encryption of model details.

A protected library typically consists of parts that are open, and other parts that need protection. Protected parts may require different degree of information hiding, for example

- Preventing unauthorized modification of models (but viewing is unrestricted).
- Parameters and documentation are visible, but model structure and equations are protected.
- The model is regarded as a "black box". Only model connectors and the icon are available to the user.

The other aspect of library protection is to ensure authorized use. In this case, any use of the library is controlled by options in a license file. A special license is also needed to *make* protected libraries in order to prevent unauthorized distribution.

Acknowledgements

The authors would like to thanks all users of Dymola, who through their suggestion have directly influenced the development of Dymola.

This work was in parts supported by the European Commission under contract IST-199-11979 with Dynasim AB under the Information Societies Technology as the project entitled "Real-time simulation for design of multi-physics systems".

References

CVS: http://www.cvshome.org/

Elmqvist, Hilding, Sven Erik Mattsson and Hans Olsson (2002): "New Methods for Hardware-in-the-loop Simulation of Stiff Models", Modelica 2002, Modelica Association.

Microsoft (1999): Microsoft Windows User Experience, Microsoft Press.

Modelica Association (2002): "Modelica — A Unified Object-Oriented Language for Physical Systems Modeling", Language specification version 2.0, January 30, 2002.

Tiller, Michael, Paul Bowles, Hilding Elmqvist, Dag Brück, Sven Erik Mattsson, Andreas Möller and Hans Olsson (2000): "Detailed Vehicle Powertrain Modeling in Modelica", Modelica 2000, Modelica Association.

Tiller, Michael (2001): Introduction to Physical Modeling with Modelica, Kluwer Academic Publ.

VRML (1997): "Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language (VRML) — Part 1: Functional specification and UTF-8 encoding", International Standard ISO/IEC 14772-1:1997.