

FINAL REPORT FROM THE PROJECT:

MODULAR SIMULATION TOOL FOR VEHICLE PROPULSION CONCERNING ENERGY CONSUMPTION AND EMISSIONS

(IN SWEDISH: *MODULBASERAT SIMULERINGSVERKTYG FÖR FORDONS DYNAMIK
M.A.P. ENERGIANVÄNDNING, EMISSIONER OCH RÖRELSE I FÄRDRIKTNINGEN,*
MED STÖD FRÅN SVENSKA FORDONSTEKNISKA FORSKNINGSPROGRAMMET ENLIGT
PROGRAMRÅDETS BESLUT MED DIARIENUMMER 8531-94-8701 DATERAT 1994-11-17)



SUMMARY

This project was run with the objective to contribute to the possibilities to estimate energy consumption and emissions for a system consisting of a vehicle with its driver, performing a transportation task in urban traffic. The goal was a computer simulation tool.

The tool developed consists of both a predefined hierarchically decomposed structure and basic module library. It uses new and powerful modelling techniques for dynamic systems. Object and equation orientation and hierarchically structured libraries are used to provide a powerful, yet flexible instrument. Reuse of modelling knowledge has been the fundamental criterion for the design of the simulation tool.

Modules for subsystems engines, transmission and chassis are parts of the result. Also, new modelling concepts of roads and drivers are developed. They make prediction of the outcome of realistically described transportation tasks possible and are useful for evaluating the influence of driver behaviour and transportation task on energy consumption and emissions.



This report is edited by:

Bengt Jacobson (project manager)
Machine & Vehicle Design, Chalmers University of Technology,
S-412 96 GÖTEBORG, Sweden

phone: int+46-(0)31 - 772 13 83
secretary: int+46-(0)31 - 772 13 60
fax: int+46-(0)31 - 772 13 75
e-mail: beja@mvd.chalmers.se
web: <http://www.mvd.chalmers.se>

Göteborg, June, 1997

CONTENTS

1	AN OVERVIEW	1
1.1	INTRODUCTION	1
1.2	RESULTS	1
1.3	POTENTIAL AND UNIQUENESS OF THE TOOL DEVELOPED.....	1
1.4	SIMULATION EXAMPLES	2
2	BACKGROUND, GOAL AND PROJECT ORGANIZATION.....	6
3	RESULTS	6
3.1	THE SOFTWARE "VEHPROP"	7
3.2	EVALUATION OF MODELLING TECHNIQUES AND SOFTWARE PLATFORMS	10
3.3	SYSTEM DECOMPOSITION AND LIBRARY STRUCTURE	13
3.4	DEMO EXAMPLES	16
3.5	PUBLICATIONS	17
4	FUTURE WORK.....	20
5	REFERENCES	22
APPENDIX A: SOFTWARE CRITERIA AND DYMOLA FULFILMENT		23
APPENDIX B: DOCUMENTATION AND USER INSTRUCTIONS		27
B.1	FILE VEHPROP/README/README.TXT	28
B.2	FILE VEHPROP/README/GETSTART.TXT	32
APPENDIX C: ECONOMY AND ORIGINAL PROJECT PLAN (IN SWEDISH)		39

1 AN OVERVIEW

1.1 INTRODUCTION

The transport sector contributes to one of the most serious problems of the man kind today if not controlled, i.e., the consumption of our energy sources and the pollution problems. These are both local and global problems, but they are emphasized in and near large cities. Therefore, it is important to understand and redesign transport systems in cities. The knowledge of emission generation and energy consumption of vehicles in city traffic is not quite consolidated, especially due to the transient propulsion which is typical for such driving, e.g. cold start.

This project was run with the objective to contribute to the estimation of energy consumption and emissions for a system consisting of a vehicle with its driver, performing a transportation task, which leads to a transient driving pattern. The goal was a computer simulation tool.

1.2 RESULTS

A computer tool for modelling and simulation of vehicle propulsion systems has been developed. The system consist of driver, road and vehicle at top level as shown in Figure 1.

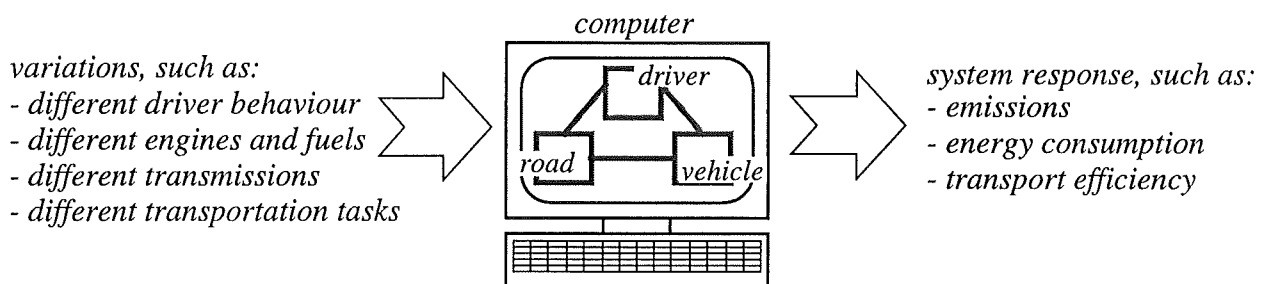


Figure 1. The tool developed and how it is to be used

The main result is presented in Section 3.1, “The Software “VehProp””, on page 7. Additionally, attention is called to four special topics:

- Section 3.2, “Evaluation of Modelling Techniques and Software Platforms”, on page 10
- Section 3.3, “System Decomposition and Library Structure”, on page 13
- Section 3.4, “Demo Examples”, on page 16
- Section 3.5, “Publications”, on page 17

1.3 POTENTIAL AND UNIQUENESS OF THE TOOL DEVELOPED

A traditional approach to vehicle propulsion simulation is inverse dynamic simulation, using a driving cycle, i.e., speed prescribed as varying in the time domain. Instead of this approach, the project has focused on true dynamic simulation (by means, e.g., a turbo lag in an engine can be considered instead of trusting in steady state maps) but also focused on more realistic driver and ambience models (by means, e.g., the road can be described in the position domain instead of the time domain). With this approach, it is possible to answer questions such as:

- How are the emissions influenced during a cold start? What is best: a cautious driver or a aggressive one? The latter one generates more emissions per time unit initially, but he reaches the warm engine state quicker, where emissions are better taken care of by the catalyst.
- The transients in fuel accumulation in the intake manifold of the engine can be studied. What is the difference in fuel consumption between operating the accelerator pedal smoothly and oscillating? The same transport task can be performed both ways.

- How does a stop, e.g., due to road construction work, influence emissions?
- How efficient is regeneration of brake energy? What control strategy is optimal?

Although the realistic ambience models strongly supports more realistic simulation of vehicles on the road, the traditional driving cycle models have an important role. The legislative requirements for certification of vehicles are, presently, expressed as driving cycles. Even if driving cycles can give a somewhat unfair, or even false, judgement of a vehicle they will probably be used for certification also in the near future. Therefore, the simulation tool also supports driving cycle based models. With the project approach, they are defined with the same interface as the more realistic ambience model, in order to support exchangeability.

The developed tool uses modern modelling and simulation techniques, i.e., equation and object oriented modelling and hybrid modelling (combined continuous and discrete dynamics). These have been found to strongly support reuse of submodels in many ways, such as, easy model library maintenance and flexibility in model topology. The software platform used, i.e., Dymola (Reference [5] and Reference [6]) has been developed considerably during the project and it has become more and more likely that this kind of modelling techniques will be more widely used in the future.

1.4 SIMULATION EXAMPLES

An example of simulation result is shown in Figure 2. The model used describes a truck with a modern turbo charged diesel engine and is visualized in Figure 3. The difference between injected fuel demand from driver and fuel actually injected by the engine ECU (Electronic Control Unit) is plotted. This fuel reduction takes the dynamic quantity boost pressure into account, i.e., the turbo charging time lag. Such a study would be impossible if only steady state engine models were used. In order to briefly describe the dynamics of the model used for this simulation, its (continuous) state variables are listed here:

- Speed of chassis
- Speed of engine crankshaft
- Speed of engine turbo shaft
- Engine boost pressure
- Driver pedal position

In Figure 4 another example is shown, where a passenger car follows a portion of the NYC (New York City) driving cycle. The engine model is visualized in Figure 5. The special feature of this model is that the dynamics of the fuel accumulation in the intake manifold is taken into account. NO_x emissions are calculated with and without this accumulation and plotted in Figure 4. The publication [Egnell, 1996a] mentioned in Section 3.5: "Publications" describes more simulation examples with otto engine models.

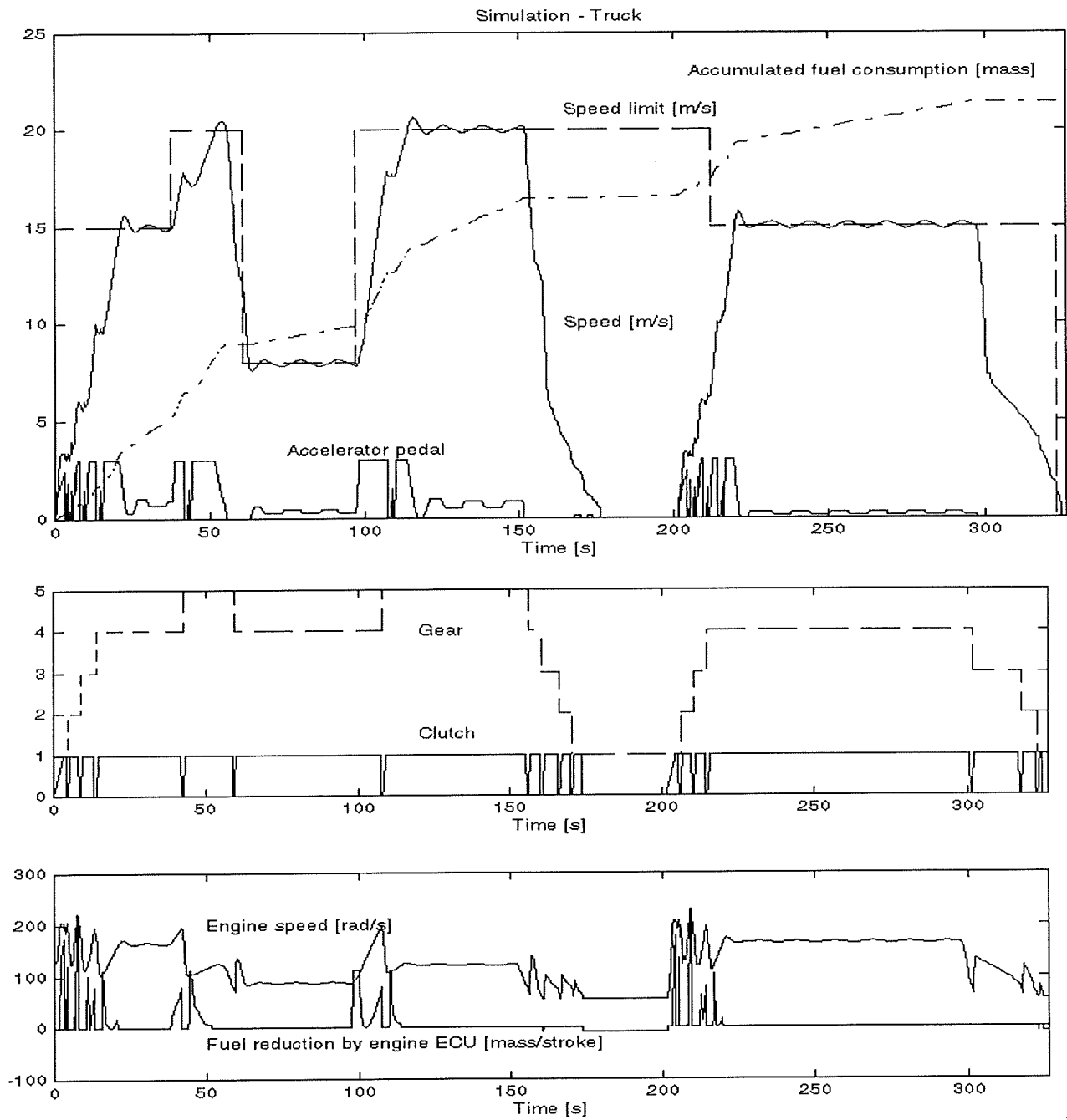


Figure 2. Example of simulation results. Truck over some minutes of driving.

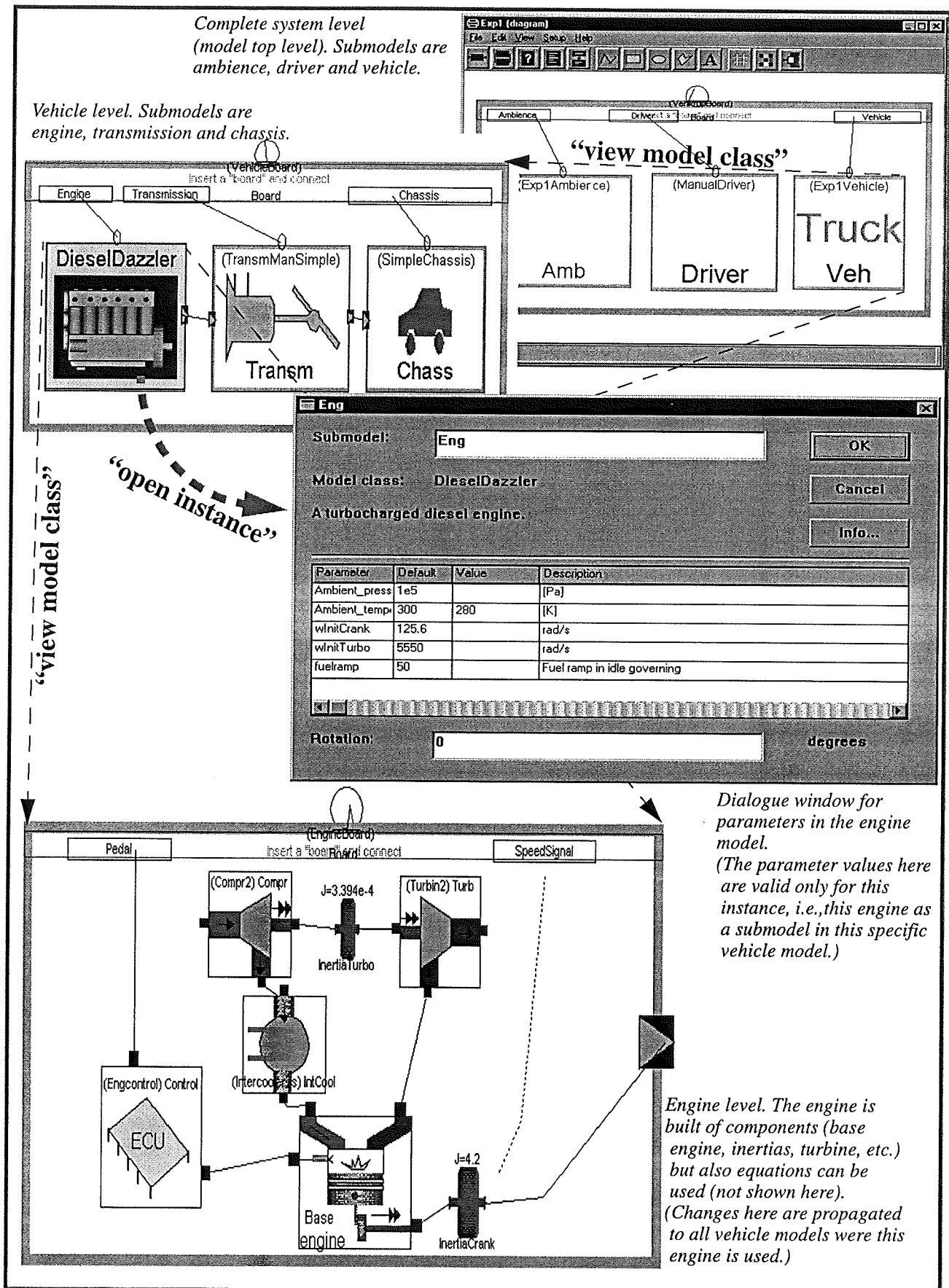


Figure 3. Example of a complete system model as visible on computer screen

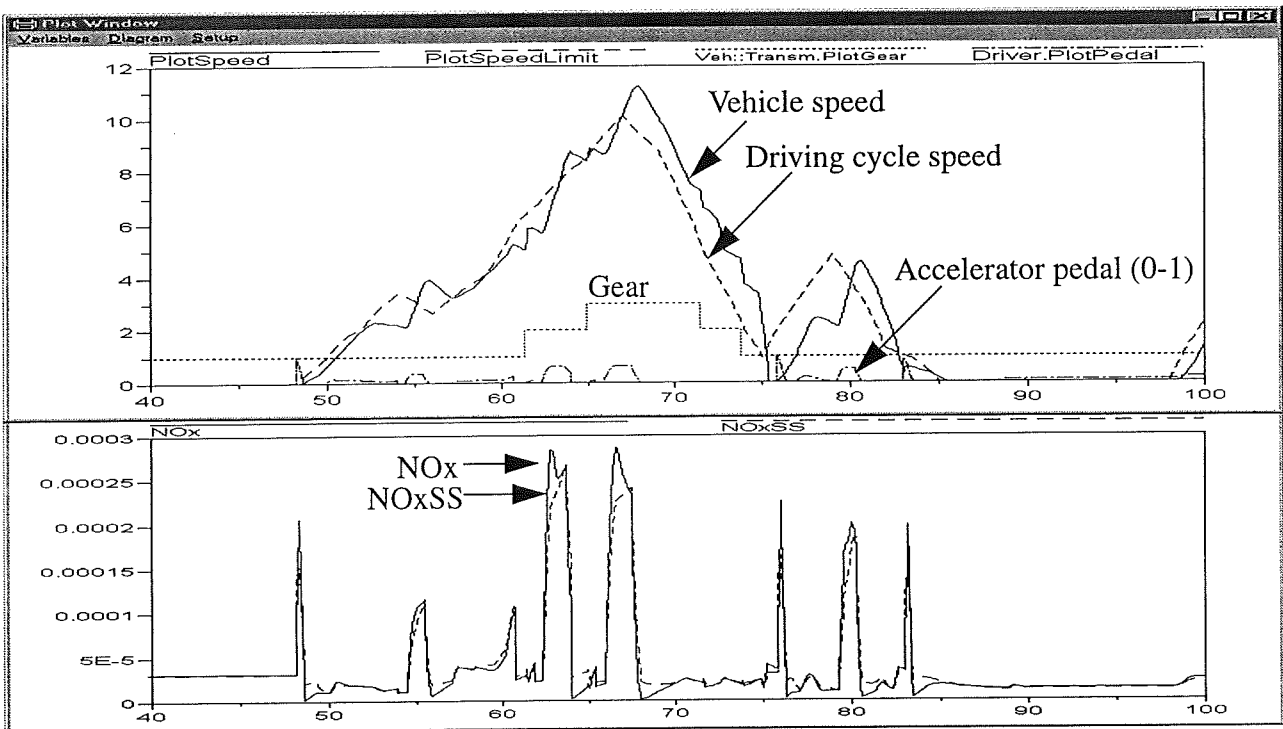


Figure 4. Example of simulation results. Passenger car without catalyst over part of NYC driving cycle. Dynamically calculated NO_x emissions (NO_x) can be compared with steady state values (NO_{xSS}) in mass unit per second, plotted over time in seconds.

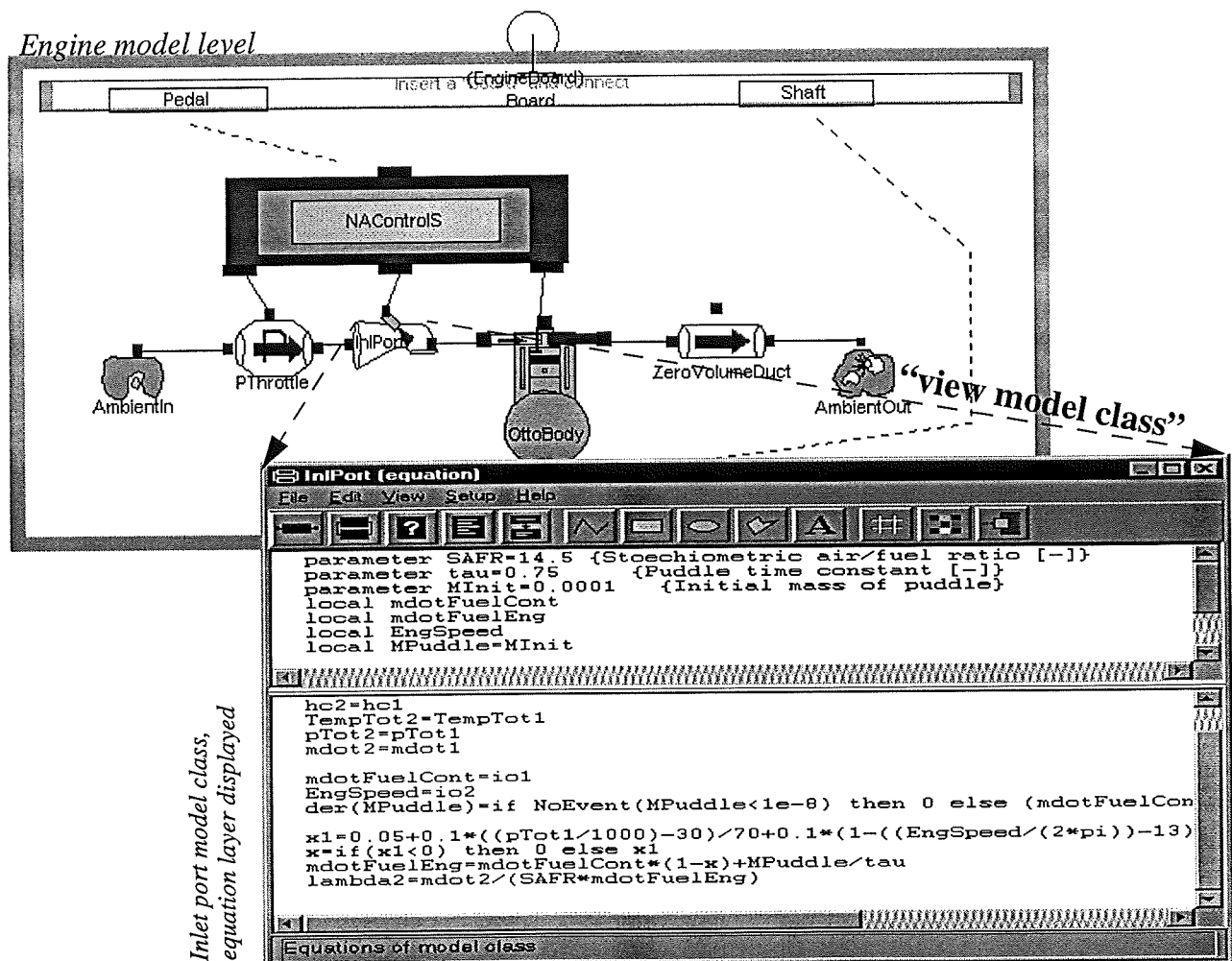


Figure 5. An Otto engine model as visible on computer screen

2 BACKGROUND, GOAL AND PROJECT ORGANIZATION

In 1994 interests in vehicle propulsion simulation coincided among the coming project participants. The field for research was defined as simulation methods and tools for vehicle propulsion, concerning energy consumption and emissions. Although the complete system vehicle-driver-ambience was to be treated, the vehicle was focused. The goal for the defined project was a simulation tool. The tool itself was the primary goal, but model development was foreseen as needed for demonstrations. Validation of these models was not declared as a goal. There were some words of honour defined:

- **Dynamics:** The tool should support models describing the dynamics (or transients), e.g., steady state engine maps would not be enough.
- **Modularity:** The tool should facilitate modularity in order to make models reusable.
- **Natural causality:** The traditional driving cycle concept, especially if combined with inverse dynamic simulation, had to be questioned. The models must allow the driver to operate the vehicle based on input from the ambience (road, traffic, etc., defining a transportation task).

The project has run during the years 1995, 1996 and the first half of year 1997. Project participants, and their tasks within the project, have been:

- **Machine and Vehicle Design**, Chalmers University of Technology, Göteborg, Sweden, (project management, transmission & chassis models)
- Department of **Heat and Power Engineering**, Lund Institute of Technology, Lund, Sweden, (otto engine models)
- Department of **Traffic Planning and Engineering**, Lund Institute of Technology, Lund, Sweden, (driver and road models)
- **Swedish National Road and Traffic Research Institute (VTI)**, Linköping, Sweden, (driver and road models)
- **Volvo Car Corporation**, (hybrid powertrain models)
- **Volvo Truck Corporation**, (truck diesel engine models)
- **Saab Automobile**, (measurement on otto engines)
- **Aspen Utvecklings AB**, (otto engine models)

The project was funded by the Swedish Board of Vehicle Technology Research (Svenska fordonstekniska forskningsprogrammet), which is administered by Swedish National Board for Industrial and Technical Development (NUTEK). These are hereby thanked for their support to reported project.

3 RESULTS

The software developed within this project is communicated on digital form, as a computer directory called "VehProp" (**Vehicle Propulsion**). VehProp (version 1.00) is the main project result and contains module libraries, demonstration examples and text files for documentation, visualized as a file browser in Figure 6. The main result is presented in Section 3.1, "The Software "VehProp"", on page 7. Additionally, attention is called to four special topics:

- Section 3.2, "Evaluation of Modelling Techniques and Software Platforms", on page 10
- Section 3.3, "System Decomposition and Library Structure", on page 13
- Section 3.4, "Demo Examples", on page 16
- Section 3.5, "Publications", on page 17

The project has, additionally to its results, contributed to widen communication channels between some Swedish actors in the field of vehicle propulsion, i.e., the project participants. Also, international contacts have been established with related activities:

- FASIMA project at TU Stuttgart, Reference [1]
- GPA (GesamtProzessAnalyse) project at TU München, Reference [2]
- Star and Pelops projects at IKA/FKA in Aachen, Reference [3]
- EngineSim (A commercial engine model library, developed in Simulink) by simcar.com, Evanston, USA, Reference [4] and Reference [8]

3.1 THE SOFTWARE "VEHPROP"

The developed software is called "VehProp" (**Vehicle Propulsion**) and is delivered as one separate file directory with the same name. The software Dymola (Reference [5] and Reference [6]) has been used as platform. It is recommended that a modern PC (e.g., Pentium 100, 30 MB RAM) with Windows 95 or Windows NT is used.

The directory structure of VehProp is shown in Figure 6. The separation in CompLib (component library) and SysLib (system library) is explained in Section 3.3: "System Decomposition and Library Structure".

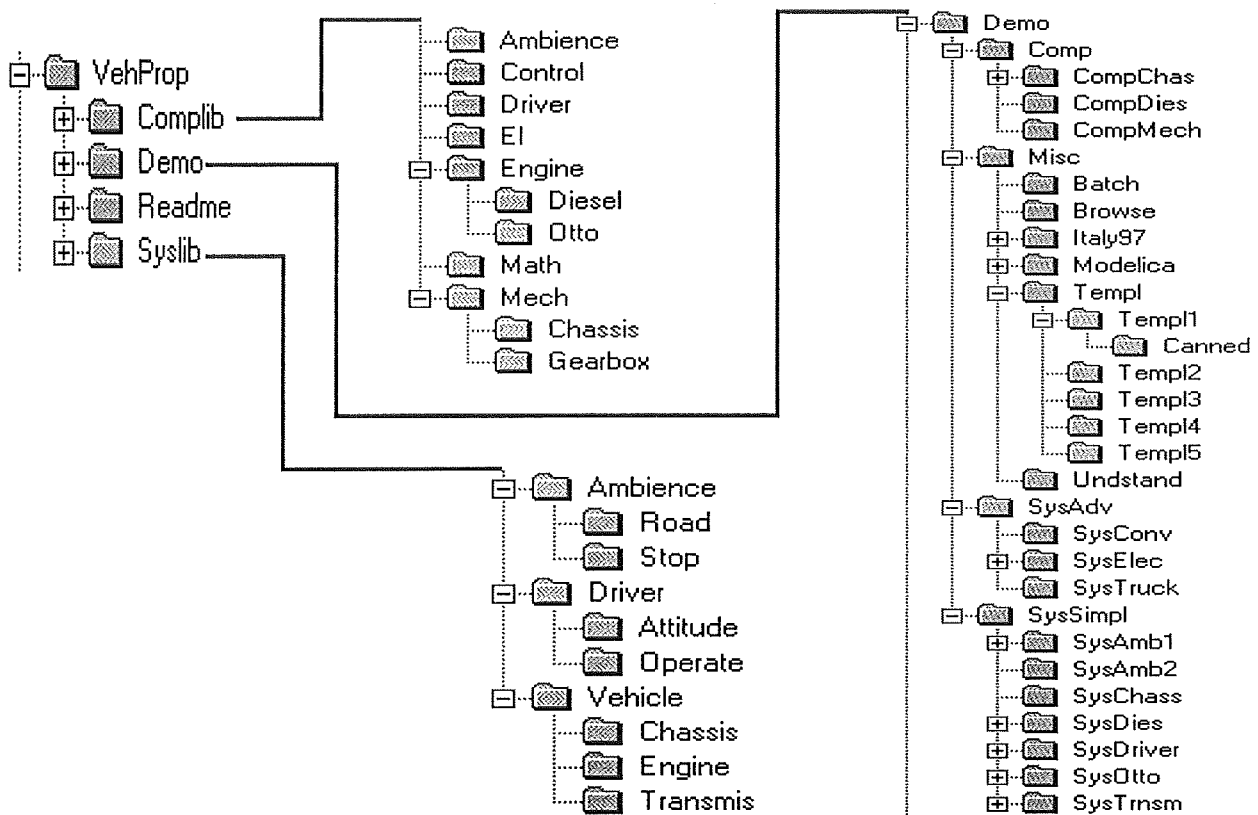


Figure 6. Directory structure in VehProp

VehProp contains approximately 5 MB data and 900 files in 8 directory levels. The file names use maximum 8 characters, to support Windows3.x. The following file name extensions are used:

- .lib: Dymola model classes (Dymola libraries and model classes)
- .dym: Dymola models
- .dyc: Dymola script files for an experiment (simulation) with a model. (Often it is suitable to have a script file xxx.dyc, which describes an experiment with the model in file xxx.dym.)
- .mat: External data file. Matrices with names on Matlab binary format. Used in Dymola models by means of the Dymola call ExternalTable.
- .m: Matlab script file
- .txt: Text file
- .ini: Dymola configuration file

3.1.1 DIRECTORY README

User instructions as .txt files, see further explanation in Appendix B: "Documentation and User Instructions".

3.1.2 DIRECTORY VEHPROP

Files of interest are:

- Dymodraw.ini, Dymosim.ini: Proposed configuration files for Dymola. Before starting Dymola from a new directory, it is suggested that copies of these files are fetched.
- wprotect.bat, unprotect.bat: DOS scripts for write protection and unprotecting, respectively, files in VehProp and Dymola.
- VehProp.lib: Top level library of VehProp
- Misc.lib: Miscellaneous library of VehProp. It contains VehProp specific cut classes ("cut" is a module interface), sublibrary icons etc.

3.1.3 DIRECTORY COMPLIB

In principal, there is only one file, a .lib file, in each subdirectory. This file contains a library and the model classes in it, e.g., see Figure 7. There might also be .mat files, which contains external data for some of the model classes in the library. Often, there is also .m files, which can be run in Matlab to create the .mat files mentioned.

```
model class (LibBase) MechComp (* library window)
submodel (Inertia) Inertia
submodel (Elasticity) Elasticity
submodel (Damper) Damper
end

model class MechBase
local PL, PR {Power at left and right cut}
cut L (wL/TL) {Left cut, w=speed, T=torque}
cut R (wR/-TR) {Right cut}
PL=TL*wL
PR=TR*wR
end

model class (MechBase) Inertia
(* description Rotating mass inertia)
(* info A rotating inertia, or flywheel, governed by:
    J*der(speed)=TorqueLeft-TorqueRight, where J=mass moment of inertia.)

parameter J=1 {Moment of mass inertia/[kg*m^2]}
parameter wInit=0 {Default initial value of speed in this instance}
parameter wScale=1 {Scale factor for integration. Approx. average speed.}
local wState=wInit/wScale

wL=wState*wScale
wR=wState*wScale
J*der(wState)=(TL-TR)/wScale
when Initial(Time) then; init(wState)=wInit/wScale; endwhen
end
```

Figure 7. From file VehProp/CompLib/Mech/Mech.lib. In general, a user never sees this textual format but uses a graphical model editor. The component library (MechComp) and a component (Inertia) with its base class (MechBase) is shown. Graphic commands etc. are removed for a better overview.

3.1.4 DIRECTORY SysLib

Files of interest are:

- `Adm.lib`: Administrative model classes, i.e., library, information class, shell class, etc. for this system structure level
- `xxx.lib`: System models, e.g., the engine model class `DieselDazzler` is stored in file `DiesDazz.lib`, see Figure 8.

There might also be a `.mat` files and `.m` files, with the same role as in `CompLib`.

```
model class (EngineShell) DieselDazzler
parameter Ambient_pressure=1e5      {[Pa]}
parameter Ambient_temperature=300   {[K]}
parameter wInitCrank=125.6 {rad/s}
parameter wInitTurbo=5550 {rad/s}
parameter fuelramp=50      {Fuel ramp in idle governing}
constant kexhp=26496
constant Rexh=288.3

output FuelRate {kg/s}
submodel (Baseengine) Baseengine
submodel (Engcontrol) Control (fuelramp=fuelramp)
submodel (Inertia) InertiaTurbo (J=3.394e-4, wInit=wInitTurbo, wScale=1000)
submodel (Inertia) InertiaCrank (J=4.2, wInit=wInitCrank, wScale=100)
submodel (EngineBoard) Board
submodel (Compr2) Compr
submodel (Turbine2) Turb
submodel (Intercooler2s) IntCool
connect Compr:Mechcut1 at InertiaTurbo:L
connect Board:ToHigher at ToHigher
connect Control:iocut1 at Baseengine:Flowcut
connect Baseengine:Mechcut at InertiaCrank:L
connect Control:IOcut1 at Board:Pedal
connect InertiaTurbo:R at Turb:Mechcut
connect Compr:Flowcut2 at IntCool:Flowcut1
connect Baseengine:Flowcut1 at IntCool:Flowcut2
connect InertiaCrank:R at Shaft
connect Baseengine:Flowcut2 at Turb:Flowcut1

Compr.p01=Ambient_pressure      {Ambient pressure, inlet [Pa]}
Turb.p04=Ambient_pressure/2+sqrt((Ambient_pressure/2)**2+ ->
    kexhp*Rexh*Baseengine.mdot3**2*Turb.T03turb)    {Turbine outlet pressure, [Pa]}
Compr.T01=Ambient_temperature   {Ambient temperature, inlet [K]}
FuelRate=Baseengine.mdottq
Board.SpeedSignal=InertiaCrank.wR
end
```

Figure 8. File `VehProp/SysLib/Vehicle/Engine/DiesDazz.lib`. Storage format of a system model, the engine model `DieselDazzler`. In general, a user never sees this textual format but uses a graphical model editor. Graphic commands etc. are removed for a better overview. Note that two of the submodels (those underlined) are instantiations of the models class `Inertia`, seen in Figure 7.

3.1.5 DIRECTORY DEMOLIB

In general each subdirectory corresponds to a demo. In principal, there is a `ReadMe.txt` file in each subdirectory, which explains the demo. See further explanation in Section 3.4: "Demo Examples".

3.2 EVALUATION OF MODELLING TECHNIQUES AND SOFTWARE PLATFORMS

The project used the software *Dymola* as platform, see Reference [5] and Reference [6].

The simulation tool of this project could have been developed from low level computer coding (C, Fortran, etc.) or medium level computer coding (scripts for *Matlab* (Reference [10]), *MatrixX* (Reference [15]), *Maple* (Reference [16]), etc.). This way was, however, dropped of resource reasons. Further on, a commercial software package has to be chosen since it is less guaranteed that freeware or university software have technical support and a development in the long term.

There are high level software, such as *ADAMS* (Reference [12]), *DADS* (Reference [13]), *SPICE* (Reference [14]), *ARLA-SIMUL* (Reference [11]), etc. All such packages were found to be very specialized on certain physical domains (3D-mechanics, rotational mechanics, electric circuits, etc.). Since, the foreseen vehicle propulsion system is of a very multi disciplinary nature (mechanics, electrics, thermodynamics, human behaviour, etc.) this way subsided for general purpose software.

General purpose software is represented by packages like *SystemBuild* (Reference [15]), *Simulink* (Reference [10]), *Simnon*, *ACSL* (Reference [9]), *Easy5*, etc. *Dymola* is another one but it is the only commercial software found that supports equation orientated and object orientated modelling of dynamic systems. These features were found to be very promising for the project aims. Therefore, there was no actual choice between different software packages but rather an evaluation of *Dymola*. (If *Dymola* would have been found unacceptable, the choice would have been among *SystemBuild*, *Simulink*, *Simnon*, *ACSL*, *Easy5*, etc.)

Figure 9 is taken from [Eriksson & Jacobson, 1997a] (see Section 3.5: "Publications"). It shows how different software can give support at different stages in a process of analysing a system. A more detailed discussion on this is given in [Eriksson & Jacobson, 1997a].

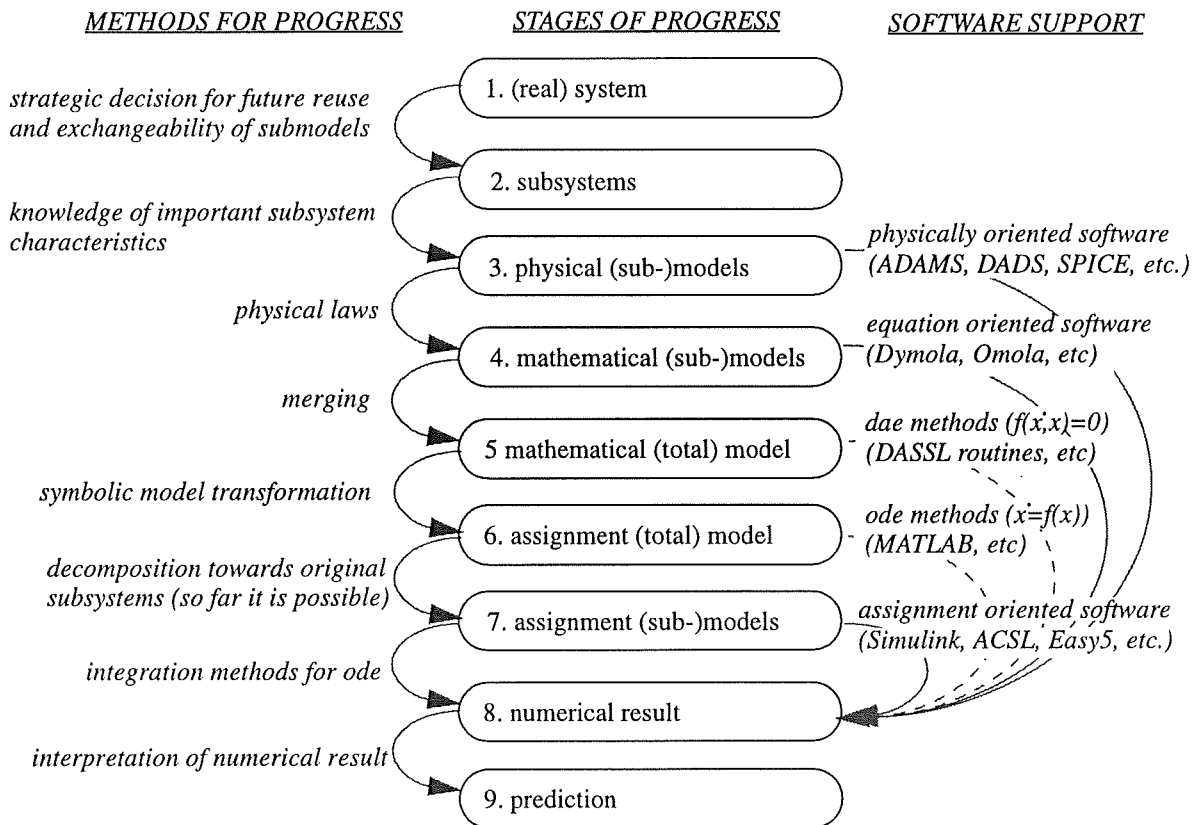


Figure 9. Suggested view of the process from knowledge of a (real) system to a prediction of how it works.

3.2.1 EVALUATION OF MODELLING TECHNIQUES

In very short terms: Equations orientation is important for system decomposition in truly physical modules and object orientation supports library maintenance. Among the publications listed in Section 3.5: "Publications", this is most thoroughly discussed in [Eriksson & Jacobson, 1997a]. Most of the points in Table 1 and Table 2 are discussed in that paper. Equation oriented modelling is to be compared with assignment oriented modelling, where all model equations has to be manually transformed to assignment statements, often graphically represented by input-output blocks. Object oriented modelling is to be compared with techniques based on copying, where new submodels are copied from a library to a model without any remaining links to the originals.

Table 1. Advantages (+) and drawbacks (-) of equation orientated modelling, as opposed to assignment oriented modelling

+	<u>Modularity</u> is supported, in the way that the same module can be used as submodel in all models. In assignment oriented modelling, a module describing a certain physical component is designed for certain surroundings. With other surroundings, the same physical component often requires development of a new module.
+	<u>Algebraic loops</u> can be avoided. In principle, groups of simultaneous equations (corresponding to "algebraic loops" in assignment oriented modelling) can be symbolically solved. In practice there will always be restrictions for strongly non-linear equations, requiring numerical iteration as in the assignment oriented case.
+	In principle, <u>constraints</u> on (continuous) state variables can be avoided, using symbolic differentiation. As example, two masses can be rigidly connected which is impossible using assignment oriented techniques. In practice there will always be some limitation in symbolic differentiation.
-	<p>It is easy to define very <u>complex or even inconsistent models</u>. When using assignment oriented modelling techniques, the user have to define input and output variables and derive all assignment statement from the equations manually, why he seldom ends up in these situations. It is common that new users experience this as frustrating rather than an extra degree of freedom in the modelling work.</p> <p><u>Solution:</u> The software have to give good diagnostics to help the user and/or the user must have a good insight in the physical problem.</p>

In general, equation oriented modelling is very suitable for physical models, since physical systems do not actually have a predefined causality. For models of causal nature, e.g., block diagram models of control systems, there is mostly no advantage in non-causal modelling.

By means of equation orientation, connections between submodels can be made very physically intuitive. E.g., a physical interface for a rotating shaft may be defined in terms of speed and torque, so that connection of three shaft ends, here numbered 1-3, generates the equations: $speed1 = speed2 = speed3$ and $torque1 + torque2 + torque3 = 0$, which corresponds to the intuitive conception of the physical connection.

Table 2. Advantages (+) and drawbacks (-) of object orientated modelling, as opposed to modelling techniques based on copying

+	Using object orientation (mainly inheritance and instantiation) it is possible to, efficiently, build and maintain two parallel library structures: <u>one component structure and one system structure</u> . If the system decomposition is changed, the component library will still be intact and vice versa. This is described more in Section 3.3: "System Decomposition and Library Structure"
+	A module have only one original, which supports <u>library maintenance</u> . All models using this module are automatically updated when the module is changed. Hereby, it is ensured that all models are updated
-	The automatic updating, just mentioned, often results in changed characteristics for the models. A user might experience this as an <u>unstable model library</u> . <u>Solution:</u> There has to be a feature to save a complete model as a stand-alone model, without links to the library. (Dymola has been equipped with such a feature.)
-	The automatic updating, just mentioned, often result in changed characteristics of the models. A user might <u>change in the model library</u> without knowing it. <u>Solution:</u> The library have to be write protected. (Dymola libraries can be write protected.)
-	A user has to be able to, starting from library models, experiment with minor changes in a model. If the library is write protected, changes are not permitted. <u>Solution:</u> There has to be a copy or duplicate feature to facilitate making working copies of models. (Dymola has been equipped with such a feature.)

3.2.2 EVALUATION OF DYMOLA

Dymola is a very new product, developed by a small company. In spite of this, Dymola was tested and found stable enough. Therefore, it became the choice of this project. More details on the project software evaluation and choice is found in:

- Appendix A: "Software Criteria and Dymola Fulfilment"
- Text files in VehProp/ReadMe/ToDynasim, Bug reports and questions to Dynasim, i.e., the company developing Dymola. Partly in Swedish.

Features of Dymola, additionally to equation and object orientation, are:

- Hybrid modelling (combined continuous and discrete dynamics). Dymola has an discrete event model syntax. Some example of advantages with the way Dymola handles discrete dynamics re given in [Eriksson & Jacobson, 1997a] and [Eriksson & Jacobson, 1997b] (see Section 3.5: "Publications").
- Dymola has an open attitude to other simulation packages.
 - * Model output on different formats. The symbolic transformation from equations to assignments can be directed to formats applicable for Simulink, ACSL, Simnon, etc..
 - * Data output on different formats. Simulation results can be written in Matlab binary data format. Hereby, it is possible to use Matlab as a postprocessor, e.g., for better plot functionality.
 - * Data input on different formats. Dymola can read data on Matlab binary data format. Hereby, it is possible to use Matlab as a preprocessor for parameters, especially useful for matrix parameters. Matlab scripts can also be used for batch jobs, such as, parameter studies, parameter optimization, linearisation, etc.
- Realtime features are not in focus for the reported project, but it is noted that Dymola is developed towards realtime applications. Dymola can generate real time code for use with Realtime Workshop of Simulink.

Some interesting development topic for Dymola are:

- Dymola will support the new international standard for symbolic modelling of dynamic systems, *Modelica*, see Reference [7]. Hereby, many efficient features will be added and competitive software companies are more likely to introduce similar tools.
- Dymola will open similar connections to the tools *MatrixX* and *Systembuild* as it has to *Matlab* and *Simulink*.

3.3 SYSTEM DECOMPOSITION AND LIBRARY STRUCTURE

Two library structure strategies are used: system libraries and component libraries. The first is an example of top-down modelling where subsystems are packed together with predefined shells (or frames) for connection to other subsystems, while the latter is an example of bottom-up modelling where basic physical components are modelled. The system models are structured in hierarchical libraries according to the structure in Figure 10, while the component models are structured mainly according to their physical domains or fields of engineering. A graphic overview of both libraries and their interaction is given in Figure 12. See also Figure 6.

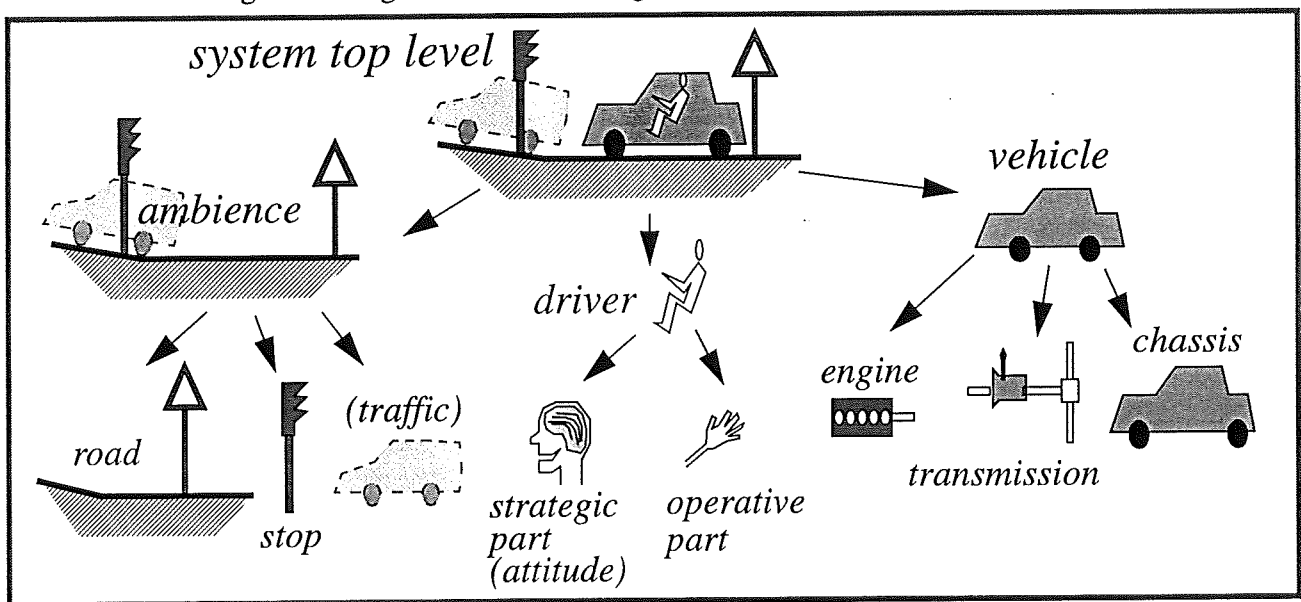


Figure 10. Proposed system decomposition. (The subsystem traffic has not been implemented in this work.)

3.3.1 BOTTOM-UP MODELLING

Components are defined on general physical bases, independently of how they interact with their surrounding. This means that the value of the component models remain even if the “top-down structure” (see Section 3.3.2) changes. Examples of components are “Inertia” (a model of a flywheel with a certain inertia) and “Volume” (a model of a gas volume). Components are defined here as models designed to be instantiated in a superordinate model, e.g., in a system as described in the next item.

3.3.2 TOP-DOWN MODELLING.

By using top-down modelling, submodel connectivity can be ensured. Figure 10 shows the hierarchical system decomposition used in this work. If the proposed structure on any level is not sufficient, custom made models can be included as long as they have the same interface to higher levels as the predefined structure.

When the decomposition is outlined, the top-down modelling work can progress. The “shell classes” defining interfaces for systems on each level are essential to this effort. The system library contains system models which fit into the proposed structure of a vehicle propulsion system, see Figure 10. All models in the system library inherit a shell class, which assures connectivity to the next superordinate

level, e.g., all models in the engine system library in Figure 12 inherits the shell class EngineShell.

As in all system theory, it can be discussed what should be called component and system models respectively. In this work, a simple rule is applied: *A system model is characterized by inheriting a shell class. Other models are component models.*

A shell class is primary used to predefine interface variables, but also module icons, output variables etc. might be subject for predefinition in a shell class. Hereby, it is possible to make all system models of a certain kind more recognizable for a user. As example, all system models have a grey frame in the icon and top level systems have variables such as PlotSpeed, PlotGear, etc.

In order to facilitate the inclusion of submodels in a system, a model class called “board” is defined for each system. Figure 11 shows an example of board model class, the VehicleBoard. As seen in Figure 11, the boards are also used to, graphically, visualize normal causality by arrows.

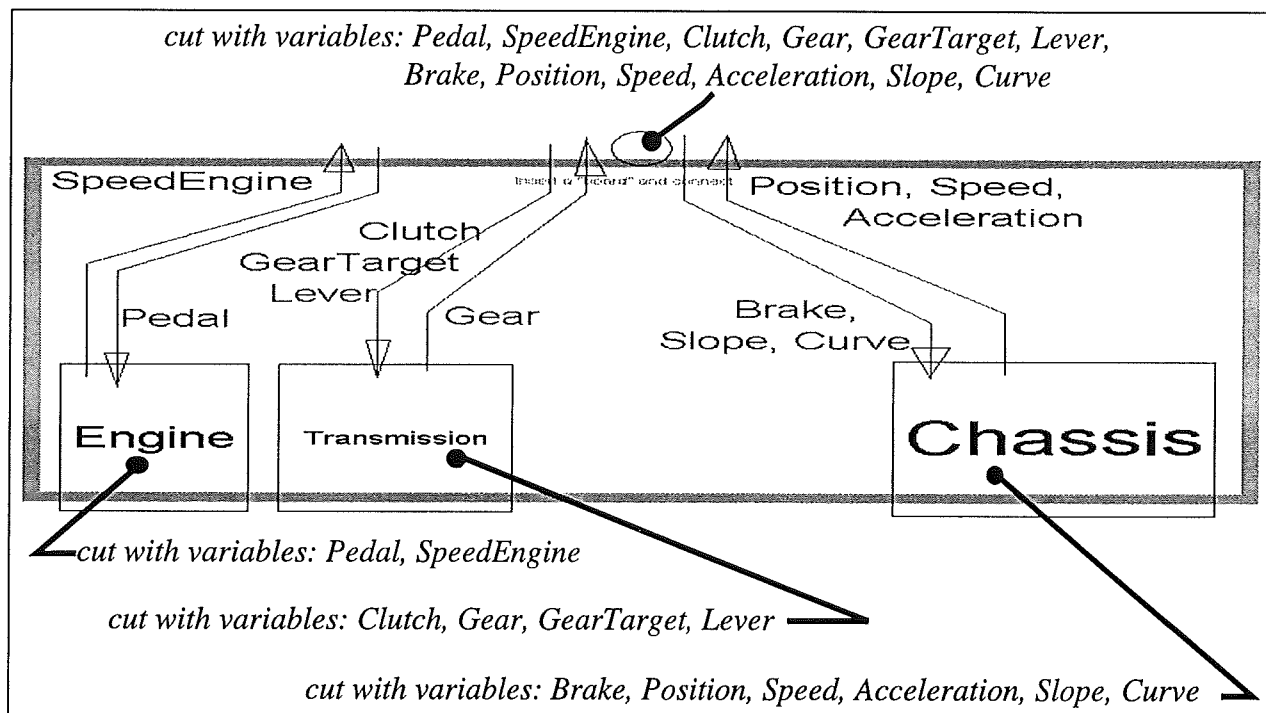


Figure 11. An example of a “board” model class, the VehicleBoard, used in vehicle system models, see e.g., upper part of the vehicle subsystem in Figure 3 on page 4. The normal signal causality is marked with arrows. (“Cut” is an interface, from which connections can be made graphically.)

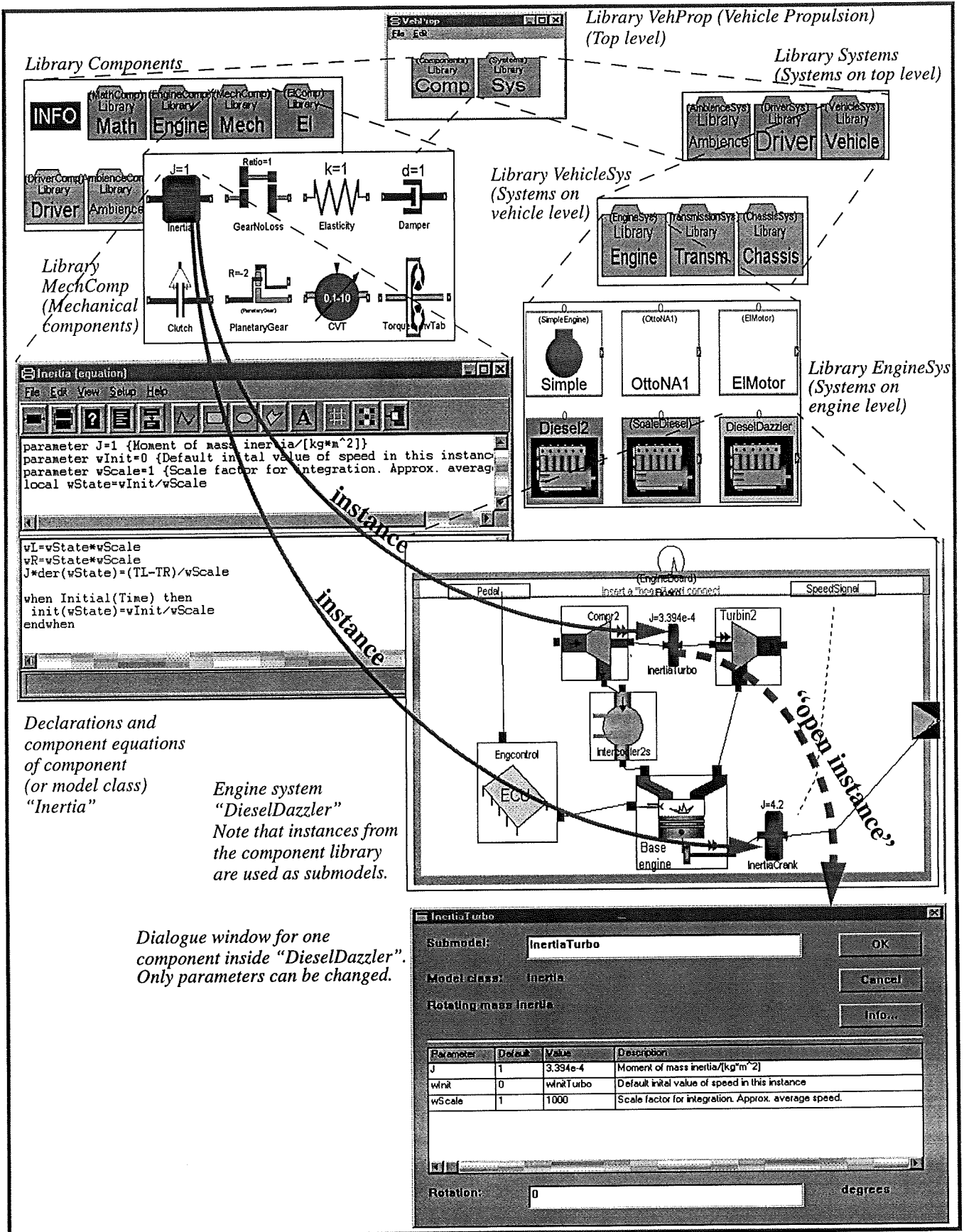


Figure 12. Component library (bottom-up modelling) and System library (top-down modelling)

3.4 DEMO EXAMPLES

VehProp contains a subdirectory "Demo", where demo examples are found. Each demo has a `ReadMe.txt` file, which describes the demo. When running a demo, it is suggested that the appropriate subdirectory is copied from VehProp/Demo to a working directory, e.g., VehPropRun. Then Dymola is started from this copy. In most cases a library or model is automatically opened and information can be reached as information stored in the model classes.

3.4.1 DIRECTORY DEMO/COMP (COMPONENT DEMOS)

These demos show some of the component models developed. The models included are not complete system models (driver, ambience, vehicle), but smaller models.

- **CompChass** (Chassis components)
Examples from [Tony Sandberg, 1996], see Section 3.5: "Publications".
- **CompDies** (Diesel engines)
Examples from [Jonas Karlsson, 1996] and [Berglund & Karlsson, 1997], see Section 3.5: "Publications".
- **CompMech** (Mechanical components)
Small models, using components from CompLib/MechLib, which contains drivetrain components, such as rotational inertias, torsional springs and clutches. Mechanical systems with ideal dry friction models and constraints are contained.

3.4.2 DIRECTORY DEMO/MISC (MISCELLANEOUS DEMOS)

- **Italy97**
Models used in [Eriksson & Jacobson, 1997a] and [Andersson & Jacobson, 1997], see Section 3.5: "Publications". This is the only example where the "optimization criterion" is used. The criteria is implemented as a separate model class, called `Penalty`, placed at system top level.
- **Batch**
A small system where the speed of an engine is regulated. A regulator parameter is optimized for fastest torque step response without overspeeding the engine. The optimization is carried out with a Matlab script. The objective is to show how batch jobs can be defined by means of Matlab.
- **Browse**
Starting from a copy of this directory just opens the VehProp library from the top level. It should be used for browsing VehProp.
- **Modelica**
Here two stand alone models are stored (called *Large* and *Small*). There are also `.mod` files, which are automatically generated stand alone models on Modelica format, see Reference [7]. Browsing these files in a text editor gives a hint of the future Modelica format.
- **Templ** (Template prepared for five complete systems)
This directory has five identical complete models, using the simplest version of systems on the lowest level from VehProp. It is supposed to be used as a template for developing new model classes and models. It is prepared for comparison of five vehicle propulsion concepts. There is a Matlab m-file with plotting instructions. See also demo `Templ/Templ1` (below).
- **Templ/Templ1** (Template prepared for one complete system)
This directory has a complete model, using the simplest version of systems on the lowest level from VehProp. It is supposed to be used as a template. The newly introduced Dymola command `DuplicateModelClass` makes it easy to generate complete system models as copies of an existing one. Hereby, the demo `Templ` (see above) has become less important: `Templ` is now a good starting directory for an arbitrary number of complete systems.
- **Templ/Templ1/Canned**
This demo contains a stand-alone (canned) version of model in `Templ/Templ1`, generated from

Templ/Templ1 by means of the Dymola command SaveTotalModel. Due to updates in VehProp, the canned version is no longer identical to Templ/Templ1.

- **Undstand**

A stand-alone demo concerning drive train models (rotational mechanics). The demo was originally developed for a student coarse in automotive transmissions. The models are build in two versions: one using physically oriented modelling and one using block oriented modelling.

3.4.3 DIRECTORY DEMO/SYSADV (COMPLETE SYSTEMS -- ADVANCED MODELS)

In order to show how complete vehicle propulsion systems of various kinds can be modelled and simulated the following demo examples are given. The models use advanced models in all subsystems, why they are quite slow to simulate, i.e., typically approximately realtime on a modern PC.

- **ConvCar** (Conventional Car)

This model describes a car with an otto engine and a manual fixed ratio transmission.

- **SysTruck**

This model describes a truck with a diesel engine and a manual fixed ratio transmission.

- **SysElec** (Cars with electrical propulsion components)

Two system models with different electrically driven cars and one with a hybrid propulsion system (combustion engine combined with energy storage in electric battery).

3.4.4 DIRECTORY DEMO/SYSSIMPL (COMPLETE SYSTEMS -- SIMPLE MODELS)

- **SysAmb1** (Complete models with different ambience models)
- **SysAmb2** (Complete models with different ambience models)
- **SysChass** (Complete models with different chassis models)
- **SysDies** (Complete models with different diesel engine models)
- **SysDriver** (Complete models with different driver models)
- **SysOtto** (Complete models with different otto engine models)
- **SysTrnsm** (Complete models with different transmission models)

3.5 PUBLICATIONS

The following publications are all, in some extent, results of scientific work within reported project. (Other references are listed in Chapter 5: "References".)

Jacobson, 1995	Jacobson, Bengt. On Vehicle Driving Cycle Simulation, <i>International Congress and Exposition</i> , Detroit, Michigan, USA, February 27 - March 2, 1995, SAE report number 950031
Dirk Fehre, 1995	Fehre, Dirk. <i>Modelling of the Transient Behaviour of Torque Converters</i> , Student project report at Machine & Vehicle Design, Chalmers University of Technology, S-412 96 GÖTEBORG, Sweden, 1995.
Jacobson& Berglund, 1995	Jacobson, Bengt and Berglund, Sixten. Optimization of Gearbox Ratios Using Techniques for Dynamic Systems, <i>International Truck & Bus Meeting & Exposition</i> , Winston-Salem, North Carolina, USA, November 13-15, 1995. SAE report number 952604
Eriksson & Jacobson, 1996	Eriksson, A. and Jacobson, B., "A Modular Model for Gear Shifting in Manual, Fixed Ratio Transmissions", <i>AVEC '96, International Symposium on Advanced Vehicle Control</i> , Aachen, Germany, June 24-28, 1996, pp. 985-1000
Andersson, 1996	Andersson, Jan. Optimum Control Strategies for Propulsion Systems in Vehicle Simulation, <i>International Symposium on Advanced Vehicle Control</i> , Aachen, Germany, 24-28 June, 1996.

Jacobson, 1996	Jacobsson, B., "Dynamic Modeling of Vehicle Propulsion Systems Using the Software Dymola", <i>CESA '96 IMACS Multiconference Computational Engineering in Systems Computations</i> , Vol. 2, Lille, France, July 9-12, 1996, pp. 964-969.
Tony Sandberg, 1996	Sandberg, Tony. <i>Dynamic Models of Vehicle Chassis and Wheels in an Equation Oriented Environment</i> , Master thesis, Machine & Vehicle Design, Chalmers University of Technology, S-412 96 GÖTEBORG, Sweden, 1996-08-23.
Jonas Karlsson, 1996	Karlsson, Jonas. <i>Dynamic Models of Diesel Engines in an Equation Oriented Environment</i> , Master thesis, Machine & Vehicle Design, Chalmers University of Technology, S-412 96 GÖTEBORG, Sweden, 1996-10-sax.
Eva Ericsson, 1996	Eriksson, E., "Att mäta bilars körmönster", Thesis for Licentiate of Engineering, Department of Traffic Planning and Engineering, Lund Institute of Technology, Lund, Sweden, 1996. In Swedish. (Approximate title in English: <i>How to Measure Driving Patterns of Cars</i>).
Egnell, 1996a	Egnell, Rolf. <i>Otto-motorer...Simulink</i> , Department of Heat and Power Engineering, Lund Institute of Technology, Lund, Sweden
Egnell, 1996b	Egnell, Rolf. <i>Otto-motorer...Dymola</i> , Department of Heat and Power Engineering, Lund Institute of Technology, Lund, Sweden
Andersson & Jacobson, 1997	Andersson, J. and Jacobson, B., "A Study of Control Strategies in Hybrid Vehicle Propulsion Systems Using Dynamic Simulation", Accepted for presentation at the <i>30th International Symposium on Automotive Technology & Automation</i> , Firenze, Italy, June 16-19, 1997.
Berglund & Karlsson, 1997	Berglund, S. and Karlsson, J., "A Modular Diesel Engine Toolbox for Studies of Charging and Control System Influence on Emissions and Performance", Accepted for presentation at the <i>30th International Symposium on Automotive Technology & Automation</i> , Firenze, Italy, June 16-19, 1997.
Eriksson & Jacobson, 1997a	Eriksson, A. and Jacobson, B., "Method for Comparison of Powertrain Concepts Using Dynamic Simulation", Accepted for presentation at the <i>30th International Symposium on Automotive Technology & Automation</i> , Firenze, Italy, June 16-19, 1997.
Eriksson & Jacobson, 1997b	Eriksson, A. and Jacobson, B., <i>Modular Modelling and Simulation Tool for Evaluation of Powertrain Performance</i> , Machine & Vehicle Design, Chalmers University of Technology, S-412 96 GÖTEBORG, Sweden, 1997.
Eriksson, 1997	Eriksson, Anders. <i>Simulation Based Methods and Tools for Comparison of Powertrain Concepts</i> , Thesis for degree of Licentiate of Engineering, Machine and Vehicle Design, Chalmers University of Technology, S-412 96 GÖTEBORG, Sweden, 1997-06-06.
Stenlås, 1997	Stenlås, Ola. <i>Chemical Model of Catalysts for Combustion Engines</i> (preliminary title), Master Thesis, Department of Heat and Power Engineering, Lund Institute of Technology, Lund, Sweden, Autumn 1997
Andersson, 1997	Andersson, J., <i>Models of Components for Electric Vehicle Propulsion, Implemented in Dymola</i> , (preliminary title), Machine and Vehicle Design, Chalmers University of Technology, S-412 96 GÖTEBORG, Sweden, Autumn 1997

[Jacobson, 1995]

Discussions to compare inverse dynamic (quasi-stationary) analysis with full dynamic analysis. Cases where quasi-stationary analysis fails, explained in mathematical and engineering terms. Model example.

[Dirk Fehre, 1995]

A student work where a dynamic model of a hydrodynamic torque converter is developed. The modelling follows the steps: physical model, mathematical model, assignment model and implemented model. The implementation is made in the simulation tool Simulink, but the equations on the mathematical model step is fitted directly for implementation in Dymola.

[Jacobson& Berglund, 1995]

An application example for vehicle propulsion simulation. The gear ratios for a truck gearbox are optimized for a certain transport task. Energy consumption and performance is weighted together. Models are implemented and simulated in Simulink.

[Eriksson & Jacobson, 1996]

A conceptual design of driver-vehicle interaction concerning manual gearshifting described. Models of manual transmissions implemented in Dymola are shown.

[Andersson, 1996]

Method and examples of how the ultimate control of transmissions can be found. The method is based on "dynamic programming" and optimizes the transmission control over a whole transport task, in a computational efficient way. This work is only partly, financed by the reported project.

[Jacobson, 1996]

Fundamental considerations on using Dymola techniques for modelling vehicle propulsion and how Dymola supports model library structure and maintenance.

[Tony Sandberg, 1996]

Models of wheel and chassis in Dymola. Somewhat more detailed than needed for energy and emission simulation. E.g., weight distribution between wheel axles, slip in both longitudinal and lateral direction, centrifugal forces in curves, etc. are modelled. Of special interest for modelling techniques are connectable modules for car and wagon.

[Jonas Karlsson, 1996]

Model of a turbocharged diesel engine for a heavy truck, implemented in Dymola. Physical modularity achieved for base engine, turbine, compressor, intercooler, etc. Resulting in a frequently used engine model, called DieselDazzler, in the project simulation tool.

[Eva Ericsson, 1996]

Discussion of methods to measure driving patterns. Measurements from the city Lund is analysed and presented. This thesis work is only to a minor extent, financed by the reported project.

[Egnell, 1996a]

Otto engine model implemented in Simulink.

[Egnell, 1996b]

Otto engine model implemented in Dymola. Based on [Egnell, 1996a].

[Andersson & Jacobson, 1997]

Describes a energy buffering powertrain for a passenger car and how it is implemented in Dymola. The mechanics of the powertrain is rather simply modelled, but the principles are good enough for the more advanced control strategy which is implemented. The control strategy is implemented by means of "Petri Net" techniques and placed as a submodel at the vehicle level, i.e., besides the submodels of engine, transmission and chassis.

[Berglund & Karlsson, 1997]

A short version of [Jonas Karlsson, 1996]. Some model development are added, such as emission models for the engine.

[Eriksson & Jacobson, 1997a]

Describes the driver and ambience models developed in reported project. The implementation in Dymola is not in focus but the concepts behind the models.

[Eriksson & Jacobson, 1997b]

Describes the simulation tool of the reported project. Focus on modelling techniques (equation and object oriented modelling) and motivates why these features are useful. It also shows some of the implementations in Dymola.

[Eriksson, 1997]

The thesis comprises three papers, [Eriksson & Jacobson, 1996], [Eriksson & Jacobson, 1997a] and [Eriksson & Jacobson, 1997b] and an summarizing text.

[Stenlås, 1997]

Models of a catalyst in Dymola. One model is a very detailed model, based on basic chemical formulas. This model is very computational expensive and is best fitted for studies of short phenomena in the catalyst itself. Another model is produced as a spin-off. It is also based on chemical formulas but less detailed and reasonably calculation efficient for ordinary studies of vehicle, driver, road-systems.

[Andersson, 1997]

Models of electric machine (motor and generator), battery, and hybrid propulsion control system in Dymola. The electric machine is special in starting from physical laws, which makes the same model class be used for motor and generator. The battery model is rather simple. Some demo examples are shown, two electric cars and one hybrid car. All with driver and ambience models.

4 FUTURE WORK

VehProp was developed as a tool, why the primary future work would be to **use** the tool for studies of vehicle propulsion. However, a software is never finally developed. It can subsequently be adopted to new techniques, why a secondary work could be to **develop** and maintain VehProp. A suggestion is that projects using VehProp are carried out but each project should be as open-minded as possible to cooperate with the others, in order to (maybe) release a common updated version of VehProp.

Reasons for such cooperation can be:

- The research group is kept together (and maybe enlarged)
- The tool is updated (especially to support the new model format Modelica, see Reference [7])
- Model exchange between projects

Presently, there are two projects planned:

- Since the reported project not had the aim to validate the models, there is a need for such validation. A work where the model parameters are calibrated against measurements on a real vehicle, driver and transport task is suggested. Such a project is visualized as *Validation of Models* in Figure 13.

The ambience models developed in the reported project do not take traffic interaction into account. However, for urban driving such interaction is very relevant and there are ideas for how traffic models should be included in the developed ambience models, see [Eriksson & Jacobson, 1997a] in Section 3.5: "Publications". Such an extension might be included in the planned project

Validation of Models in Figure 13.

More information: *Börje Thunberg, Swedish National Road and Traffic Research Institute (VTI), S-581 95 LINKÖPING, Sweden*

- Control system design with realtime applications (Rapid Prototyping, Hardware-in-the-loop, etc.). New control systems, at vehicle level, are in focus. There is a special methodology concern in using equation oriented and object oriented modelling for both simulation and generation of real time code. Such a project is visualized as *Integrated Powertrain Control* in Figure 13.
More information: *Bengt Jacobson, Machine & Vehicle Design, Chalmers University of Technology, S-412 96 GÖTEBORG, Sweden, E-mail: beja@mvd.chalmers.se*

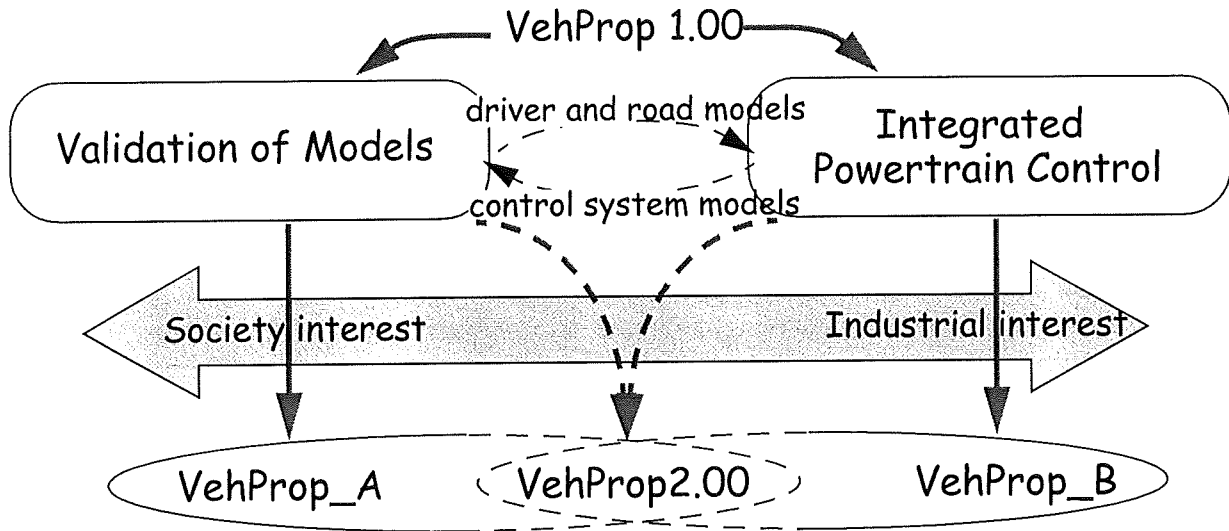


Figure 13. Planned projects and possible interaction dashed.

5 REFERENCES

(See also Section 3.5, "Publications", on page 17.)

- [1] Spörl, T., "*Modulares Fahrsimulationsprogramm für beliebig aufgebaute Fahrzeugtriebstränge und Anwendung auf Hybridantriebe*", Dissertation, Institut für Maschinenelemente, Universität Stuttgart, Germany, 1996. In German.
- [2] Woschni, G., Schwarz, C. and Zeilinger, K., "Längsdynamiksimulation von Kraftfahrzeugen mit dieselmotorischem Antrieb", *MTZ*, 56 (1995), 4 pp. 206-211.
- [3] Helling, J. et al., "Antriebsstrukturen für Kraftfahrzeuge - Zielsystem, Entwicklungswerkzeuge, Verbesserungspotentiale", 4. *Aachener Kolloquium Fahrzeug- und Motorentechnik '93*, Aachen, Germany, October 5-7, 1993, pp. 711-732.
- [4] Rubin, J.Z., Munns, S.A. and Moskwa, J.J., "The Development of Powertrain System Modeling Methodologies: Philosophy and Implementation", *SAE Paper No. 971089*, 1997.
- [5] Elmquist, H., "*Dymola User's Manual*", Dynasim AB, Lund, Sweden, 1994.
- [6] <http://www.dynasim.se>, information on the modelling and simulation software Dymola.
- [7] <http://www.dynasim.se/modelica.html>, information on an international standard of dynamic modelling languages, Modelica.
- [8] <http://www.simcar.com/products.htm>, information on the engine model library EngineSim
- [9] <http://www.mga.com/>, information on the simulation software ACSL from MGA
- [10] <http://www.mathworks.com/>, information on the software Matlab and Simulink from Mathworks
- [11] Laschet, A., *Simulation von Antriebssystemen*, Springer-Verlag 1987 (Theoretical base for software ARLA-SIMUL from ARLA Maschinentechnik GmbH, Kürten, Germany)
- [12] <http://www.adams.com/>, information on the simulation software ADAMS from Mechanical Dynamics
- [13] <http://www.cadsi.com/dads.html>, information on the simulation software DADS from CADSI
- [14] <http://sss-mag.com/spice.html>, information on the simulation software SPICE
- [15] <http://www.isi.com/>, information on the software MatrixX and SystemBuild from Integrated Systems
- [16] <http://www.maplesoft.com/>, information on the software Maple

APPENDIX A: SOFTWARE CRITERIA AND DYMOLA FULFILMENT

In Table 3 - Table 6, criteria for a software platform is listed along with how Dymola fulfil them.

- Table 3, “Software criteria and Dymola fulfilment -- Practical aspects”, on page 23
- Table 4, “Software criteria and Dymola fulfilment -- Openness”, on page 24
- Table 5, “Software criteria and Dymola fulfilment -- Modelling facilities”, on page 25
- Table 6, “Software criteria and Dymola fulfilment -- Tools”, on page 26

The selection of criteria is based on comparison with other general purpose simulation software, such as SystemBuild, Simulink, ACSL, etc. Dymola is also discussed in Section 3.2, “Evaluation of Modelling Techniques and Software Platforms”, on page 10.

Table 3. Software criteria and Dymola fulfilment -- Practical aspects

Aim or criterion	Dymola fulfilment
The software platform should be in use for approximately 10 years	Dymola is developed by a small company. However, Dymola uses new promising techniques and has a good potential to stay in business for a long time. Of special interest is the new international standard Modelica, see Reference [7], which will make modelling efforts less dependent of a specific software dealer.
Reasonable licence fees	Most licence fees, including Dymola, are reasonable compared to develop and maintain own in-house code.
Quick start for a beginner	Dymola requires rather much understanding of the user. There are a lot of separate log files, configuration files etc. However, with well prepared demo examples, it is possible to get the desired quick start.
Support, Manuals, On-line help	The company Dynasim, which develops and trade Dymola, is known as very competent and seems to have a sincere wish to contribute to the world wide front line of modelling techniques. Hereby, technical support is given as good as possible, with respect to the modest size of the company. There is a User Guide (Reference [5]) on paper. Dymola also has (almost) the whole User Guide as on-line help.

Table 4. Software criteria and Dymola fulfilment -- Openness

Aim or criterion	Dymola fulfilment
Support on different computer platforms	Dymola is primarily developed for PC with Windows95 or WindowsNT and good processor performance. Other platforms are supported (presently SUN and Silicon Graphics), even if new Dymola versions often are delayed some months.
Real time applications	Dymola can generate code for Simulink (on Simstruct format). Hereby, realtime applications has been made with Dymola models. Dymola is developing towards more build-in support for generation of realtime code
Pre- and postprocessing of simulation in general purpose numerical software, e.g., Matlab.	Dymola interfaces Matlab by reading data from Matlab binary format files and by storing simulation result on files with same format. This, together with Matlab scripts shipped with Dymola, Matlab can be used as a batch script format. Dymoview is the plot tool of Dymola. It is rather primitive. However, Matlab can be used for plotting. A development of Dymoview or an extended connection to other software such as, Matlab or MatrixX is desired.
Incorporation of external routines in Fortran, C, etc.	Dymola can incorporate C-functions in models.
Openness to other simulation software	Dymola generates model output on different format, such as Simstruct (for Matlab/Simulink), ACSL, Simnon.
Openness for the user	The model storage format and the generated c-code for simulation are fully open and readable as text files. The built in routines, such as integration routines, are not delivered on readable format.
Models on closed form for distribution without revealing model structure.	A compiled Dymola model can be created and distributed.

Table 5. Software criteria and Dymola fulfilment -- Modelling facilities

Aim or criterion	Dymola fulfilment
Possible to create stand-alone models (without links to the original library)	Dymola has a SaveTotalModel command that makes a stand-alone model (canned model), contained in one separate file.
Arbitrary description of relations between variables. E.g., equations, table interpolation, neural net, etc.	Dymola support equations (including multi-dimensional matrix formalism) and table interpolation (both in Dymola internal parameter matrices and matrices from an external file). For special features, such as neural nets, the user has to arrange his own C-functions.
Information/documentation in model definitions	Dymola syntax allows description (one line) for each parameter defined and a model class information layer (several lines).
Warnings when a variable passes the range within which a model is valid.	Dymola syntax has no such build in feature. One work-around is to write an own c-function, error, and call it conditionally: "x=if y<yMax then 7 else error("message")". Another work-around is to construct equations like "x=if y<yMax then 7 else 1/0", which makes the simulation stop by failure when y becomes >yMax.
Model parameters	The parameter handling in Dymola is well developed. Default values can be given, parameters can be propagated between hierarchic model levels, etc.
Time and state event modelling, e.g., stick-slip phenomena for dry friction.	Dymola has a strong support for time and state events. It is based on a high level syntax with if- and when-statements.
Petri nets, State transition nets, or similar	Dymola has a library with Petri Net blocks, which is efficiently supported by the state event handler.
Model library support	Dymola very strongly support that the user defines their own model libraries. There are special model classes, library classes, and the object oriented techniques allows efficient library maintenance.
Possibility to define an arbitrary default value for an unconnected port (cut, interface, etc.).	Dymola sets the value zero on an unconnected cut variable of through-type. (Through-type variables are summed to zero when connected, e.g., connecting cut(speed1/force1) to cut(speed2/force2) generates equations force1+force2=0). Other default values cannot be defined.
Data file for external tables	There has to be one external data file with a specific name, dsdata.mat, in current directory. In order to guarantee that data from VehProp library is used, a file must be copied from VehProp or a Matlab script for merging data from VehProp can be run. A better solution might be: In the model definition, default file name and path should be given as string parameters. Then the default data could be the data from VehProp library.
Instantiating in base class	Submodels instantiated in a base class cannot be connected in an inheriting class. This is a major drawback, but will be dealt with in next Dymola version (Dymola4.0).

Table 6. Software criteria and Dymola fulfilment -- Tools

Aim or criterion	Dymola fulfilment
Time efficient simulation	Since Dymola uses compiled code the simulation is relatively fast. However, when developing models, the compilation time can be disturbingly long.
Trimming tools (for finding suitable initial conditions)	Dymola has no trimming tools. On the other hand, the non-causal models allows a user to change calculation order by adding constraint equations for state derivatives, e.g., " $\frac{dx}{dt}=0$ " and solving the system for x . Hereby, a suitable initial value of x is found and the system can be solved for x through integration of $\frac{dx}{dt}$.
Tools for linearization and eigenvalue analyses	Dymola has no such tools. A linearization tool will be added in next Dymola version.
Efficient and various integration methods available.	Dymola has several methods implemented. Also, they are parameterized. Parameters are, e.g., maximum order of integration allowed, maximum number of iterations in each time step, etc.
Tools for Control design, System identification, Signal processing and Optimization	Dymola has no such tools. The closest is that Dymola is shipped with a Control block library (similar to Simulink) and a Petri Net library.
Documentation support	Dymola has a command to generate model documentation on html format. The command can be given from any library, model or model class. The generated html document contains graphical icons (bitmap format) and links from submodel instantiations to model class definition.
Debugging tools	The debugging features of Dymola is mainly output to a log file, <code>dslog.txt</code> . Here, the simulation can be followed. Additionally, the models is checked for consistence when read from file and when the models are partitioned, e.g., by the symbolic transformation.
Iteration tolerance	Cannot be controlled (except for the inconvenient way to edit the generated c-file, <code>dsmodel.c</code>) It is desired that the iteration tolerance could be controlled from the Dymola menu, like integration tolerance.
Write protection	The only way to write protect a model library is to use the file write protection of the operating system. This is too fragile, since write protection information cannot be retained when transferring a model library to a new computer. It is desired that Dymola could administrate the write protection.
Model library administration -- Where is a model class used?	Using Windows 95, there is a feature to find files where a certain string, e.g., a model class name, appears. Hereby, the information can be found. It is desired that Dymola could administrate the search.

APPENDIX B: DOCUMENTATION AND USER INSTRUCTIONS

Sources of information on Dymola:

- Dymola User's Guide, Reference [5]
- Dymola on-line help available during a Dymola session (approximately the same information as Reference [5])
- Text files shipped along with Dymola (for installation, news, known problems, etc.). E.g., `Dymola/ReadMe.txt`
- Dynasim homepage on internet, Reference [6]
- There is a Dymola mailing list. See <http://www.Dynasim.se/malinglist>.
- There is, or will soon be, a Dymola users forum at internet. Probably somewhere at the web site: <http://www.Dynasim.se>.
- There is an international Dymola User's Group. Meetings will be announced to licensed users.

Sources of information on VehProp:

- `VehProp/ReadMe/ReadMe.txt` This file is printed Section B.1 on page 28 and contains four parts:
 - 1) Installation instructions for VehProp,
 - 2) List of information sources,
 - 3) Overview of VehProp library structure,
 - 4) Recommendations for models in VehProp.
- `VehProp/ReadMe/GetStart.txt` A text file with some exercises to start with VehProp. It is printed in Section B.2 on page 32.
- Text files in `VehProp/ReadMe/ToDynasi` Here bugs and problems are listed. Some of them has a solution or work-around described. It might be useful to browse these files. Some text in Swedish.
- `VehProp\ReadMe\Troubble.txt` -- General trouble shooting - some tips
- `VehProp\Demo\ReadMe.txt` -- Overview of demo examples
- `VehProp\Demo\...\ReadMe.txt` -- Presentation of each demo example
- `VehProp\ReadMe\News.txt` -- News and known problems with the present version of VehProp
- `VehProp\ReadMe\Adresses.txt` includes addresses of the participants in the project.
- Model classes in VehProp have their "information layer" and "parameter descriptions", which can be reached during a Dymola session.

B.1 FILE VEHPROP/READMe/READMe.TXT

```
=====
= Main ReadMe-file for Vehicle Propulsion Library, VehProp                      =
=====
= The VehProp library was developed by the project:                            =
=                                                                              =
= "Modular Simulation Tool for Vehicle Propulsion,                            =
=  concerning Energy Consumption and Emissions"                               =
=                                                                              =
= (In Swedish:                                                                =
=  "Modulbaserat simuleringsverktyg för fordons dynamik                    =
=   m.a.p. energianvändning, emissioner och rörelse i färdriktningen",       =
=  med stöd från Svenska Fordonstekniska Forskningsprogrammet enligt        =
=  programrådets beslut med diarienummer 8531-94-8701 daterat 1994-11-17)    =
=====
```

This file contains four parts:

- INSTALLATION INSTRUCTIONS FOR VehProp
- INFORMATION SOURCES
- OVERVIEW OF VehProp LIBRARY STRUCTURE
- RECOMMENDATIONS FOR MODELS IN VehProp

```
=====
INSTALLATION INSTRUCTIONS FOR VehProp
=====
```

- 0) You would probably get less problem the "better" computer you use.
 Minimum recommendations:
 PC, Pentium100 MHz, RAM30MB, 100 MB free space on hard disc, Windows95 or WindowsNT
- 1) Install Dymola. See instructions from Dynasim AB.
 It is recommended that Dymola3.0f is used.
 (For more information on Dymola, see internet: <http://www.Dynasim.se>)
- 2) Run the delivered file VehProp.exe. It is a self-extracting file.
 The directory "c:\VehProp" will be created.
 (It is possible, but not recommended, to chose another directory path.)
- 3) Add the your location of VehProp to DYMOLAPATH in AutoExec.bat.
 If VehProp is located as c:\VehProp, the following line
 should be present in your Autoexec.bat:
 set DYMOLAPATH=c:\VehProp
- 4) Using Matlab scripts shipped with Dymola
 =====
 If you would like to use Matlab for pre- and postprocessing of compiled
 Dymola models, you should add the path ../Dymola/Mfiles/traj (where ... in
 most cases should be c:) to the path defined for Matlab in the matlab file
 ../matlabrc.m (where ... in most cases should be c:/Matlab).
- 5) Using external data files on Matlab binary format with Dymola
 =====
 Some models in VehProp uses external data files. Then there have to be a file
 called dsdata.mat in Dymola current directory. The demos are prepared with
 such files. However, if you would like to update the dsdata files by means of
 Matlab, you should add the path ../VehProp to the path defined for Matlab in
 the matlab file ../matlabrc.m (where ... in most cases should be c:/Matlab).
 Hereby, you can access the external data files stored in VehProp from matlab.

It is then easy to create a matlab script, such as
VehProp/Demo/SysSimpl/SysAmb2/WriDsd.dat.m, which updates the file dsdata.mat,
with
the newest version of data files in VehProp.

6) UPDATES VIA INTERNET

=====

It is possible to get updates of Dymola via Internet: www.Dynasim.se/update.
The only file not provided via internet is the licens file.

7) AUTOMATIC DOCUMENTATION IN HTML FORMAT

=====

The automatic HTML-documentation feature of Dymola requires:

- * In Autoexec.bat: set DYMOLAHTML=1
- * You also need the file: Dymola\bmp2gif.exe

8) Write protection

=====

VehProp would work best if the library files were write protected. Since some model classes from the libraries from Dynasim (shipped with the software Dymola) is used in VehProp, also Dymola itself should be write protected. The only available way to write protect is to use the write protection administrated by the operative system (e.g., Windows95). However, such write protection disappears when VehProp is zipped and unzipped again during delivery. Therefore, VehProp is not write protected when delivered to you. However, two script file are delivered:

- VehProp\wprotect.bat

Click on this file in your file manager. Then, all files under c:\VehProp and under c:\Dymola will be write protected.

- VehProp\unprotect.bat

Click on this file in your file manager. Then, the write protection done by VehProp\wprotect.bat will be released.

Anyway it might be a good idea to store your own backup copy of the complete VehProp. Then you can update your VehProp easily if you happen to write in the library files.

=====

INFORMATION SOURCES

=====

LIST OF MORE TEXT FILES TO READ:

- VehProp\ZipIntro.txt -- Version identity file, view when unzipping
- VehProp\ReadMe\GetStart.txt -- some exercises to start with VehProp
- VehProp\Demo\ReadMe.txt -- Overview of demo examples
- VehProp\Demo\...\ReadMe.txt -- Presentation of each demo example
- VehProp\ReadMe\ToDynasim\...\txt --
-- some questions and answers about Dymola, some in Swedish
- VehProp\ReadMe\Adresses.txt --
-- some adresses of the participants in the project which developed VehProp
- VehProp\ReadMe\Troubble.txt -- General trouble shooting - some tips
- VehProp\ReadMe\News.txt -- News and known problems with the present version
- VehProp\ReadMe\ToDo.txt -- Some possible future updates of VehProp

OTHER SOURCES:

- Dymola Users Guide (shipped from Dynasim AB with Dymola licens)
- Dymola -- Selected publications (shipped from Dynasim AB with Dymola licens)
- Web site of Dynasim AB: <http://www.Dynasim.se>
- Final project report (from the project which developed VehProp 1995-1996)
- Publications (from the project which developed VehProp 1995-1996).
Listed in final project report.

- Personal contacts regarding VehProp, e.g. through the file:
VehProp\ReadMe\Adresses.txt

=====

OVERVIEW OF VehProp LIBRARY STRUCTURE

=====

The file names uses maximum 8 characters, to support Windows3.x. The following file name extensions are used:

- .lib: Dymola model classes (Dymola libraries and model classes)
- .dym: Dymola models
- .dyc: Dymola script files for an experiment (simulation) with a model. (Often it is suitable to have a script file xxx.dyc, which describes an experiment with the model in file xxx.dym.)
- .mat: External data file. Matrices with names on Matlab binary format. Used in Dymola models by means of the Dymola call ExternalTable.
- .m: Matlab script file
- .txt: Text file
- .ini: Dymola configuration file

VehProp has 4 important directories: CompLib, SysLib, Demo and ExtData. They will be described in the following.

- VehProp - CompLib - Control
 - Driver
 - El
 - Engine - Diesel
 - Otto
 - Ambience
 - Math
 - Mech - Gearbox
 - Chassis
- SysLib - Driver - Attitude
 - Operate
- Ambience - Road
 - Stop
- Vehicle - Chassis
 - Engine
 - Transmis
- Demo - ... (See VehProp\Demo\ReadMe.txt)

1) VehProp\CompLib -- the component directory

=====

Here, components are stored. Definition of components: "Components" are building blocks used as submodels in the "systems" on the lowest level.

In each subdirectory there is at least one file, a .lib-file with component models (only model classes, no models). Additional files should also be stored here, e.g., .mat-files for back-up of external data used in the component models. Also, corresponding .m files might be stored here for generation of the .mat files.

2) VehProp\SysLib -- the system directory

=====

Here, systems are stored. Definition of systems: "Systems" are build for fitting into the proposed structure of a vehicle propulsion system. Systems on the lowest level use "components" as submodels. Systems of higher level use lower level systems as submodels. All systems inherit a "shell class", e.g.,

EngineShell, DriverShell, etc.

In each subdirectory there is a file called Adm.lib. This is the administrative model classes, such as "Library", "Interface", "Shell", "Board", "Info", etc. Other .lib-files are also present and they each include a model class inheriting the shell class. Such a model class is here called a system.

Files for external data as in the component library might also be stored here (.mat and .m files).

3) VehProp\Demo -- the demo directory

=====

This directory should have several subdirectories, each corresponding to a demo. To run a demo, copy one of the demo directories to somewhere outside VehProp, e.g., to c:\Test. Start Dymola from this location. There are more information in ReadMe.txt files of each demo directory. See also VehProp\Demo\ReadMe.txt

=====

RECOMMENDATIONS FOR MODELS IN VehProp

=====

In general, the following recommendations is given:

- * Cuts should, preferably, be defined as physical as possible. E.g., it is recommended to use through variables ("after-slash-variables").
- * A demo should be able to start without the file Dymola.lib present
- * Info text should be written. Here, you should try to cover all problems that can occur, such as:
 - Is there need for an external table, i.e. a .mat file. If so, document the path where it can be found.
 - Does the model need any non default Dymola settings. As default we use the settings in file Dymola\Insert\Dymosim.ini. For instance, the Dymola setting "DefaultConnect on" is Dymola default.
 - Does the model need any Dymola commands (like "Differentiate" or "Variable value unknown x").
 - Name and date for who and when the model class was created
- * All model classes and models which needs special initial values (on continous or discrete state variables or iteration variables) should have suitable initial value defined by a parameter.
- * External data can be used by adding a .mat file. It is recommended that also a .m file for generation of the .mat file is supplied. Both .mat and the corresponding .m files should be stored in VehProp/CompLib or VehProp/SysLib depending on the type of model class it is used.

=====

= Bengt Jacobson =

= Machine & Vehicle Design, Chalmers University of Technology, =

= S-412 96 GÖTEBORG, Sweden =

= Phone: int+46-(0)31-772 13 83 (secretary: ...13 60) =

= Fax: int+46-(0)31-772 13 75 =

= E-mail: beja@mvd.chalmers.se =

= Web: http://www.mvd.chalmers.se =

=====

B.2 FILE VEHPROP/README/GETSTART.TXT

```
=====
= GettingStarted-file for Vehicle Propulsion Library =
= File: VehProp\ReadMe\GetStart.txt =
=====
```

First install Dymola and VehProp as described in VehProp\ReadMe\ReadMe.txt.
Then, you can do the following exercises.

EXERCISE 1,

Running av demo example and change parameters

```
=====
```

- 1) Make your own working directory. E.g., c:\VehPropRun
- 2) Copy the directory VehProp\Demo\Misc\Templ\Templ1\Canned to c:\VehPropRun.
Change directory name from c:\VehPropRun\Canned to c:\VehPropRun\Exerc1.
(This change of name is not actually important, but makes it more clear
that we should work from an OWN COPY of the demo directory.)
- 3) Start Dymola from the new directory c:\VehPropRun\Exerc1. (I.e., open the
directory in the explorer and chose Start/Run Dymola (in Windows95).)
There should pop up two windows:
 - Dymola - Dymola Modelling Laboratory (Command window)
 - Templ1 (The model we will simulate)Additionally, there are some minimized windows:
 - Dymoview
 - Plot Window
- 4) In Dymola window there is a toolbar at the top of the window. Chose the
most left tool (or, chose from menu: File/RunScript). A file browser
will pop up, from which you chose file Templ.dyc.
- 5) Wait. In the Dymola window, you will see information that the model classes
are loaded and instaciated. The model is partitioned, compiled and simulated.
On a PC Pentium166MHz 32MB RAM, you have to wait about 15 seconds.
- 6) Now open the window Plot Window. Chose variables. From the variable list you
may chose, e.g., PlotSpeed, PlotSpeedLimit, PlotPedal and PlotGear. Klick OK,
and the simulation results will be plotted. On the x-axis, there is Time in seconds.
PlotSpeed and PlotSpeedLimit is plotted in m/s. PlotPedal is the accelerator
pedal position, meassured as a number between 0 and 10. PlotGear shows the gear,
where PlotGear=1 corresponds to 1st gear, 20 to 2nd gear, etc.
-) Exercise 1 could be over now, but if you are interested in parameter
studies, please continue...
- 7) Dubble click on submodel Amb (Ambience) in window Templ. A dialog window
will pop up. Scroll down to parameter Speed1. Nothing but the default value
5 m/s is given (in the "default box"). Write instead 10 as value (in the
"value box"). Klick OK.
- 8) Now chose File/SaveAll in the Templ window. Answer yes on the question
of saving to file c:\VehPropRun\Exerc1\Templ.dym.
(The saving will take some time, because Templ.dym is a large file.
Actually Templ.dym contains a stand-alone model (or "canned" model),
where all model classes are defined in the same file. So, there is no
references to VehProp library.)
- 9) Perform a new simulation as you did in point 4 and 5.
-) Now the result should be plotted, but you are recommended to do this in
new plot window. Therefore...
- 10) Open the window Dymoview and chose View/PlotWindow. A new plot window
pops up and you may minimize Dymoview window again.

- 11) Plot as you did in point 6.
-) Now you have learned how to change parameter values by really updating the model description in its file. This was made when you choose File/SaveAll. There is a another way to change parameter values. If you are interested go on.
- 12) Try another way to change parameters: Chose the tool (or button) "p=" in the toolbar of the Dymola window. A parameter browser will pop up. Browse down to the parameter Model/Amb/Speed1 and mark this parameter. Its value (10) will then be displayed in the upper part of the parameter browser. Change the value to 15 (m/s) and klick OK.
- 13) Run a simulation again, but please try the most right tool (or button) in the toolbar of the Dymola window (or chose Simulation/Simulate from the menu).
A new simulation will be runned, and we should notice that this time, it takes shorter time, since the parameter was changed in the Dymola window and not in the Templ window, as we did in points 7 and 8. For instance, there was no need for a new compilation. In short, in points 7 and 8 we changed the parameter in the model and in point 12 it was changed only in the experiment.
- 14) Now, we can plot the new result. Let us try to plot it in the same plot window. Chose, in the plot window, SetUp/AutoErase. (Maybe you have to chose it several times to obtain that there should be no marker before the word "AutoErase".) Now plot PlotSpeed by chosing Variables/PlotSpeed in the plot window.
- 15) Chose File/Exit either from Dymola window or from any edit windows.

EXERCISE 2,

Browsing a model and changing a subsystem

=====

- 1) Do point 1, 2 and 3 in exercise 1 again, BUT with VehProp\Demo\Misc\Templ\Templ1 instead of VehProp\Demo\Misc\Templ\Templ1\Canned instead. It might be suitable to rename your own working directory with a name such as Exerc2.
- 2) Now a window called Develop will pop up. This is a library window. Klick with your right mouse button at Templ1 in this window. Select ViewClass. Then a window called Templ1 will open. This window is an edit window.
- 3) The model class Templ1 will look very much like in exercise 1, but now the model classes from VehProp (c:\VehProp) are used as submodels. (In exercise 1, only copies from VehProp were used.) You can test this by doing ViewClass on submodel Amb (in Templ1) and then on Road (in Amb). Try to save SimpleRoad (which is the model class of the submodel Road) by clicking Ctrl+s in the Road window. Dymola will answer "Cannot open file ... write protected", since you then tried to write to a file within VehProp, which should be write protected. If Dymola CAN save, you have probably forgot to write protect VehProp. Do this after instructions in file VehProp\ReadMe\ReadMe.txt. Close SimpleRoad and Templ1Ambience windows before carrying on.
- 4) Do ViewClass on Veh in Templ1. In the window Templ1Vehicle, you will then see the submodel Eng, which is of the model class SimpleEngine. SimpleEngine is a very simple engine model, so we try to change to better one.
If you like, you might view the SimpleEngine model class. When you have the edit window for SimpleEngine, chose Edit/EquationLayer (or click on the fourth tool (or button) from left at the top of the window). In the equation layer you can see the variable declarations and equations of the model class SimpleEngine. The SimpleEngine is very simple. Roughly, the torque follows a square polynom.
- 5) Open the VehProp library by chosing File/Library/VehiclePropulsionLibrary.
A library window VehProp will open. (It takes some time, library reading is logged

in window Dymola, if you want to follow it online.) This is the top level of VehProp library. You can go down to engine models by double-clicking with your left mouse button on sublibraries Sys (in VehProp), then Veh (in Systems), then Engine (in VehicleSys). Then you have the engine library with the engine models OttoNA1, OttoNA2, etc.

- 6) Select the engine in Temp1Vehicle by clicking once on it. It then becomes marked with red "handles". While it is still marked, chose Edit/ChangeSubmodel and write OttoNA1. The engine model in Temp1Vehicle will then change to an OttoNA1 instead of a SimpleEngine.
(If we had not opened the engine library of VehProp, Dymola would have had to ask for which file OttoNA1 was to be found. The correct answer is VehProp\SysLib\Vehicle\Engine\OttoNA1.lib, but that might be difficult to know...)
(There is another way to change submodel: Drag OttoNA1 from the engine library into Temp1Vehicle. Mark and delete the SimpleEngine. Connect OttoNA1 with mouse.)
(It might be a good idea to close windows Systems, VehicleSys and EngineSys now. You have probably already noticed that there can be a lot of windows opened. Learning to close or minimize windows helps!)
- (If you like, view the model class of the OttoNA1! It is much more detailed than SimpleEngine. It consists of physical submodels from the component library of VehProp. If you like, you can open the relevant component library by double-clicking on Comp (in VehProp), Engine (in Components), Otto (in EngineComp).)
- 7) Now, try SaveAll and RunScript and Plot (See exercise 1, point 8, 9 and 6). The simulation will not work! You will be told that "Dymosim could not find the data file dsdata.mat".
- 8) There is obviously something special with the new engine model. Try right mouse button on OttoNA1 in window Temp1Vehicle. Chose Info. The information window will tell you (among other things) that the data file VehProp\CompLib\Engine\Otto\dsdata.mat is needed. So, copy it to your work directory.
- 9) Try to simulate again. Either by running the script again or, faster, as described in excersise 1, point 13. The simulation should work now and you should be able to plot the results.
It might be your interest to view the results from the emission model. If so, plot the variable Veh::OttoNA1.NOx. (This variable is lifted up to top level of the engine, which can be read in the info text of OttoNA1. Point 8 tells you how to view this info.)

EXERCISE 3

Browse the VehProp library

=====

- 1) Do point 1, 2 and 3 in exercise 1 again, BUT with VehProp\Demo\Misc\Browse instead of VehProp\Demo\Misc\Temp1\Temp1\Canned. It might be suitable to rename your own working directory with a name such as Exerc3. The VehProp top level library opens up.
-) The points 5 and 6 contains some brief browsing of the system and component libraries in VehProp library. In this exercise some ideas with the library structure will be more clearly pointed out.
- 2) Open (by double-clicks) VehProp--System--VehicleSys--EngineSys--DieselDazzler. All engines in the engine library "inherits" the base class EngineShell. This is why they look very much the same. The idea of this inheritance is to force all engine models to be exchangeable in a superior model class. (A superior model class for an engine is a vehicle. And all vehicles inherits "VehicleShell", and so on...). If you want a tidier screen, close System, VehicleSys and EngineSys.
- 3) Open (by double-clicks) VehProp--Components--EngineComp--DieselComp. Close EngineComp. Also open VehProp--Components--MechComp. Close Components. Now, you are supposed to observe (or believe in) that all submodels in the DieselDazzler engine are "instanciations" or "instances" of model classes in the component library of VehProp. E.g., there are a compressor model class

"Compr2" and an inertia model class "Inertia" used. The inertia class is instantiated twice, as the instances "InertiaTurbo" and "InertiaCrank", but with different parameter settings for the mass moment of inertia.

-) In conclusion, the VehProp library has two branches, the system library and the component library. The system library contains models for use in vehicle propulsion models, packed in a way that they fit together. The component library contains models of more general physical nature for use in any type of models, not only vehicle propulsion models. The systems uses the components as submodels. It might be noted that components are not the only way to model a system - also equations can be used in parallel. See, e.g., the model class SimpleEngine with no components at all or OttoNA1 with some connections made by equations.
- 4) The third sublibrary in VehProp top level library, Misc, contains some miscellaneous model classes. Just open it and browse it briefly.

EXERCISE 4:

Build a new model

=====

- 1) Do point 1, 2 and 3 in exercise 1 again, BUT with VehProp\Demo\Misc\Templ instead of VehProp\Demo\Misc\Templ\Templ1\Canned. It might be suitable to rename your own working directory with a name such as Exerc4.
- 2) Now a window called Develop will pop up. This is a library window. Open sublibrary MyTempl by double-clicking wt it with your left mouse button. Here you see five models, Templ1, ..., Templ5. All these are simple models, for use as templates when defining new models. View model class Templ1.
- 3) Open VehProp system library, by choosing File/Library/SystemLibrary. A library window called Systems will open up.
- 4) In the same way as exercise 2, point 6, you should now change the model class of the following submodels in Templ1:
 - * submodel Veh/Eng (has model class SimpleEngine, change to ScaleDiesel)
 - * submodel Veh/Transm (has model class SimpleTransm, change to TransmManSimple)
 - * submodel Driver/Att (has model class SimpleAttitude, change to BlockAttitude)
 - * submodel Driver/Op (has model class SimpleOperate, change to ManOperate)
 - * submodel Amb/Road (has model class SimpleRoad, change to RoadTab.
Answer yes on the question "Discard such parameters...")
 - * submodel Amb/Stop (has model class SimpleStop, change to StopTab)
- 5) Now, try SaveAll and RunScript (C:\VehPropRun\Exerc4\Templ1.dyc) and Plot (See exercise 1, point 8, 9 and 6).
The simulation will not work! You will be told that "Dymosim could not find the data file dsdata.mat". Copy VehProp\CompLib\Engine\Diesel\dsdata.mat to VehPropRun\Exerc4 (cf exercise 3, point 8). Now the simulation will run.
- 6) Plot PlotSpeed and PlotSpeedLimit. The simulated driving issue will be poorly fulfilled.
E.g., in the beginning of the simulation, the speedlimit is 10 m/s, but the vehicle only achieves approximately 3 m/s.
-) In the following points 7, 8 and 9, some trouble shooting in the model will be shown. It points out that this kind of driver and ambience models often are sensitive to parameter settings.
- 7) By some parameter setting in the submodels, a more realistic simulation result can be obtained. In order to explain the models, a reasoning is first made: Plot Veh/Transm/PlotGear. The driver do not shift to higher gear than 1st gear! Browse the model again. Double-click with left mouse button on Templ1/Driver/Op displays the parameter setting for the operate part of the driver. The parameter SpeedUp is 350 rad/s, which means that the driver do not shift up until the engine speed becomes higher than 350 rad/s. Plot Veh::Eng.Speed_. It never becomes higher than 350 rad/s (the engine only reaches 200 rad/s). This must be the reason why the

driver never shifts! Plot Driver.PlotPedal. It shows that the driver pushes the (accelerator) pedal to maximum. Why does not the engine reaches higher speeds. Read the info of submodel Temp11/Veh/Eng (hold down right mouse button and chose Info). Reading the info, we note that this is a scales version of a large truck diesel engine. This must be the reason why the engine does not reaches higher speeds. OK, so our first try could be to change the scaling of the engine: Double-click with left mouse button on model Temp11/Veh/Eng. This action displays the parameters of the engine model. Give the value 3 to SpeedScale and value 0.1 to TorqueScale. SaveAll and RunScript again. Now, PlotSpeed follows PlotSpeedLimit better.

- 8) The plotted speed dips to zero at time 15-20 seconds. It seems like the vehicle does not follow the speedlimit very well here. However, a stop is defined here. Plot Amb::Stop.Stop. It shows a step from stop number 1 to 2. This means that the driver here passes a stop. Change independent variable by open the Dymoview window and chose Setup/IndependentVariable and PlotPosition. Then plot PlotSpeed and Amb::Stop.Stop. Change parameter Driver::Att.StopMargin from 5 to 0 m. Run a new simulation and plot. Then you see that the driver is modelled in a way that he can miss a stop and go on, if he is not given margins large enough.
- 9) Now, change parameter Driver::Op.SpeedEngDiseng from 150 to 50 rad/s. Try a new simulation. The simulation will probably fail, since the driver does not release the clutch in time resulting in that the engine decelerates and dies. To plot the simulation results from a failed simulation, you have to open window Dymoview and chose File/OpenResult and file dsres.mat.
- 10) With Driver::Att.StopMargin = 5 and Driver::Op.SpeedEngDiseng = 150, the simulation result is descent. Note that you can plot the fuel consumption [kg/s] by plotting Veh::Eng.FuelRate. If you would like to plot the accumulated consumption and consumption per distance and also lift the information to the model top level, do as follows: Open the equation layer of window Temp11 (i.e., chose View/EquationLayer). In the upper part of the equation layer you should declare:
 output FuelRate, FuelAccum, FuelPerDist
 In the lower part you should define:
 FuelRate=Veh::Eng.FuelRate
 der(FuelAccum)=FuelRate
 FuelPerDist=if Position>1 then FuelAccum/Position else 0
 {If is used to avoid division by zero.}
 Do SaveAll and RunScript. Now, you can plot FuelRate, FuelAccum and FuelPerDist. (Maybe you noticed that we didn't need to declare the variable Position. This is because it is defined in all models inheriting the base class, VehPropShell, which is the case for Temp11.)
-) You have now developed a new model by changing submodels in Temp11. Often, one would like more than one model, for comparison etc. This is why Temp12, ..., Temp15 is present in the library Develop. (This directory and file structure is also suitable when several persons should cooperate in model development. Then person 1 and 2 have their own directory, Temp1\Temp11 and Temp1\Temp12. Then they both start Dymola from Temp1, but are responsible for model classes in each subdirectory.) The models Temp12, ..., Temp15 could be changed in similar ways, maybe by using other transmissions (automatic, CVT, etc.). Here, it should be noted that comparisons between closely related systems can be made with the same model, but using different ".dyc files". Above, we have used the .dyc file Temp11.dyc. If we, e.g., just want to change gear ratio of 5th gear from 5 to 4.5, we could make a copy of Temp11.dyc and add the row:
 parameter Veh::Transm.RatioVec_1 = 4.5
 somewhere before the line: simulate
- 11) If you are satisfied with your model (Temp11) you can now build a "stand-alone model" (or "canned model"). A good idea is to make a new directory for this model. Use the file manager of your computer to make a subdirectory Exerc4/Canned. Then choose File/SaveTotalModel in the Temp11 window. Give the file Canned/TotMod.dym.
- 12) Test the canned model by, first, exit Dymola. Then start Dymola again from your directory Excer4/Canned. Do File/Open TotMod.dym from the window Unnamed. You should know, that the version of Temp11 now is completetly stand-alone from

the VehProp library. All model classes opened are stored in the file TotMod.dym. Any changes can be made without disturbing VehProp library and any changes in VehProp library will not affect this canned model.

EXERCISE 5:

Define your own model classes

=====

- 1) Do point 1, 2 and 3 in exercise 1 again, BUT with VehProp\Demo\Misc\Templ\Templ1 instead of VehProp\Demo\Misc\Templ\Templ1\Canned instead. It might be suitable to rename your own working directory with a name such as Exerc5.
- 2) Open edit window Templ1/Veh. ViewClass on submodel Transm, i.e., view the class SimpleTransm. We will develop a new transmission starting from a duplicate of this model class. Choose File/New/DuplicateModelClass from window SimpleTransm. Give it an own name, maybe MyTransm.
Save your new model class in a file by choosing File/SaveModel from window MyTransm. A suitable file name might be MyTrans.lib. MyTransm is now an exact duplicate of SimpleTransm.
- 3) Mark the submodel Transm in the present vehicle model. Chose Edit/ChangeSubmodelClass and write MyTransm. It is now important to note the change in the model class Templ1Vehicle. You are suggested to do File/SaveModel on Templ1Vehicle. Dymola will ask you to accept storage in file Exerc5\MySys.lib. Accept it, but note that, if, you had made a change in a vehicle model class of the VehProp library, you had not offered to write to the file (unless you had forgot to write protect the VehProp files). The proper way is then, NOT to release the write protection but, to duplicate also the vehicle model class to a new model class name (e.g., MyVehicle) and saved it in a new file (e.g., MyVehic.lib).
- 4) Now we will do some changes in MyTransm. First delete the everything in diagram layer and equation layer. You will not be able to delete the frame in the diagram layer, since it is inherited from the base class: TransmShell.
- 5) Now, open the VehProp component library, by choosing File/Libraries/ComponentLibrary. Open sublibrary Mech. Drag two components into MyTransm (diagram layer): one TorqueConvTab and one GearNoLoss. Before going further, you are suggested to give your two components new names. (This is not really necessary, but emphasizes the difference between model class and submodel.) Double-click on TorqueConvTab. In the dialog box, change the model NAME to "Conv" (but remember, it is still of the model CLASS TorqueConvTab). In the same manner, change the name of GearNoLoss to "gear". (Note that you should not use "Gear", since "Gear" is a variable name in MyTransm. Variable "Gear" is defined in the base class of MyTransm, i.e., TransmShell.)
- 6) Connect, by mouse and its left button:
 - * From the "cut" at the left side of MyTransm to left cut of Conv
 - * From right cut of Conv to left cut of Gear
 - * From right cut of Gear to right cut of MyTransm
- 7) Double-click on Gear. Give the parameter Ratio the value R. (We could have given it a numerical value, but with R, we will show the principal of parameter propagation.) The value R should be defined in MyTransm. Open the equation layer of MyTransm. Write the following line in the upper part:


```
parameter R=5 {Ratio of gear transmission}
```
- 8) Switch to the icon layer of MyTransm. Draw something using the drawing tools in the toolbar of MyTransm. (E.g., open the palette, mark everything in the present icon and chose a new colour.)
- 9) Close window MyTransm. Double-click on Transm in Templ1Vehicle. Note that you now can give another value than 5 for your parameter R, e.g., 7. Also note your comment of the parameter "Ratio of gear transmission".
- 10) SaveAll and RunScript Templ.dyc. It will not work. Dymola will complain that there are more variables than equations. Add the line "Gear=1", to the lower part of the equation layer of MyTransm. (This is not easy to realize, but might be credible if double-clicking

MODULAR SIMULATION TOOL FOR VEHICLE PROPULSION CONCERNING ENERGY CONSUMPTION AND EMISSIONS

on the top cut of MyTransm. Among other cut variables we find the variable "Gear". Gear is a variable with normal causality OUT from the transmission. Therefore, we should define it somehow, e.g., as identical to 1.) Now, the simulation will work.

-) Above, we defined a "system", i.e., a model class inheriting a shell class (in this case a transmission, inheriting TransmShell). Below we will define a "component", i.e., something of general physical character, useful also for other simulations than vehicle propulsion simulations.
- 11) Chose File/New/ModelClass from any edit window. Give the name "Visco", base class "MechBase" and library "MyComp1". Do SaveModel from window Visco, into the new file Visco.lib. (The name of the base class is not obvious, but might be found suitable if you browse the base class of other mechanical components, such as GearNoLoss. Neither the library class is obvious, but it is suggested in the Info in the window Develop.) We are about to define a simple model of a visco coupling, i.e., a piecewise linear damper.
- 12) Write the following lines in the upper part of the equation layer of Visco:
parameter d1=2, d2=20
parameter wCrit=50
Write the following line in the lower part of the equation layer of Visco:
TL=if abs(wL-wR)>wCrit then d1*(wL-wR) else d2*(wL-wR)
TR=TL
We use the inherited variables wL/wR (speed at left/right end) and TL/TR (torque at left/right end).
Draw some simple icon in the icon layer of Visco.
- 13) Open sublibrary MyComp1 from the library Develop. Drag a copy of Visco to your model class MyTransm. Open up (delete) the connection to the right of Gear and connect the Visco between Gear and the right cut of MyTransm.
- 14) SaveAll and RunScript again. You can plot the usual PlotSpeedLimit and PlotSpeed, but note also the variables Veh::Transm::Visco.PL and Veh::Transm::Visco.PR. These are the mechanical power of left and right shaft on the Visco. They are defined in the base class MechBase, and were hereby inherited to Visco.

EXERCISE 7:

RUNNING THE VehProp DEMOS

=====

Read the file VehProp\Demo\ReadMe.txt. There is a list of demos and a short description.

This GettingStarted intructions were developed by:

=====

= Bengt Jacobson	=
= Machine & Vehicle Design, Chalmers, Sweden	=
= Phone: int+46-(0)31-772 13 83	=
= Secretary: int+46-(0)31-772 13 60	=
= Fax: int+46-(0)31-772 13 75	=
= E-mail: beja@mvd.chalmers.se	=

=====

APPENDIX C: ECONOMY AND ORIGINAL PROJECT PLAN (IN SWEDISH)

Table 7. Från NUTEK rekvbirerade medel (kSEK). Sorterade efter kostnadsslag

Kostnadsslag	budgeterat	rekvirerat
löner	1 980	2464
datorkostnader	240	129
resor	180	169
konsulter, mjukvara	640	248.5
administrativa pålägg	960	989.5
sum	4 000	4 000

Table 8. Industribidrag (kSEK)

företag	enligt projektplan	upparbetat	innehåll
Volvo Car Corporation	1 200	400	provbil
		700	modeller av elkomponenter
		100	mantid
Volvo Truck Corporation	1 200	1 000	motormodell
		200	mantid
Saab Automobile	1 200	100	motormap
		300	bänkprov
		150	CVS prov
		50	simulering
		500	motormap, steady state
		100	mantid
Aspen Development	400	100	datorprogram
		100	transient korrigering
		200	mantid
summa	4 000	4 000	-

Table 9. Från NUTEK rekvbirerade medel (kSEK). Sorterade efter budgetpost

Budgetpost varifrån rekvirerats	budgeterat	rekvirerat
mvd, Chalmers	2 000	2 000
kraft&värme, LTH	1 200	1 200
trafikteknik, LTH (inkl. VTI)	400	203
styrgrupp	400	597
summa	4 000	4 000

Table 10. Från NUTEK rekvbirerade medel (kSEK). Sorterade efter mottagare

Mottagare av rekvirerade medel	budgeted	rekvirerat
mvd, Chalmers	2 000	2 488
kraft&värme, LTH	1 200	1 243
trafikteknik, LTH	100	123
VTI (formellt del av Trafik)	300	146
Styrgrupp	400	0
summa	4 000	4 000

Bilaga: Projektplan för projektet

Modulbaserat simuleringsverktyg för fordons dynamik m. a. p. energianvändning, emissioner och rörelse i färdriktningen

Sammanfattning

Fordons transienta körförlopp i stadstrafik kan idag inte analyseras tillfredsställande. Projektet avser därför att utveckla ett simuleringsverktyg för analys av systemet fordon-förare-väg-trafik, vad gäller fordonets dynamik m. a. p. energianvändning, emissioner och rörelse i färdriktningen. Grundidén är att arbeta modulbaserat — olika företag/personer ska kunna utveckla egna och använda andras moduler. Projektet kommer i första hand att resultera i ett programskal, med väldefinierat inre protokoll, men även ett grundbibliotek med färdiga moduler. Mjukvaran görs utvecklingsbar och användarvänlig för att nå acceptans hos användarna (industri, myndigheter, forsknings- och undervisningsanstalter) under en längre period.

Introduktion

Ett fordon ska samverka med förare, väg och övrig trafik. Omvärlden påverkas och ställer krav på hela situationen. Främst gäller kraven avgasemissioner och energianvändning i stadstrafik, men även fordonsprestanda och personsäkerhet. Det är ofta önskvärt att studera situationen som ett tidsförlopp. Exempel på sådana studier är simulering av körning enligt körcykel, start, växling, styrmanöver samt krock.

Fordonsindustri, myndigheter, forsknings- och undervisningsanstalter har intresse av att utföra simuleringar enligt ovan. Utan samarbete uppstår nackdelar som: dubbelarbete, icke jämförbara modeller, kommunikationssvårigheter samt risk att befintlig spetskompetens inte utnyttjas i modellens alla delar.

Ett datorbaserat simuleringsverktyg för fordonssimulering skulle kunna utgöra ett gemensamt fundament för alla som sysslar med denna typ av analys. För att vinna verklig framgång måste det dessutom vara modulbaserat. Att täcka in alla typer av fordonssimuleringar är dock knappast praktiskt möjligt. En inriktning mot dynamik i färdriktningen skulle kunna vara lagom omfattande och betjänar det aktuella samhällsbehovet att optimera fordons framdrivning, speciellt i stadstrafik. Meningsfulla studier av körning i stadstrafik kräver äkta transienta analyser, vilka idag inte kan hanteras tillfredsställande. Att utveckla ett sådant datorverktyg är målet för detta projekt. Oavsett hur spridd användningen av själva mjukvaran blir, kommer gemensamma synsätt och definitioner som utvecklas att underlätta kommunikation mellan industri, myndigheter, forsknings- och undervisningsanstalter.

Mål

Projektets övergripande mål är att skapa förutsättningar för avancerade studier av fordons framdrivning, speciellt i stadstrafik. Som medel för detta ska ett modulbaserat simuleringsverktyg för fordons dynamik i färdriktningen utvecklas. Som delresultat kommer synsätt och definitioner att samordnas inom projektet. Eftersom industri, myndigheter och högskola finns representerade i projektgruppen kommer synsätt och definitioner förhoppningsvis att finna acceptans även utanför projektet.

Följande punkter definierar simuleringsverktygets egenskaper:

- ☐ Ett **gemensamt format** för modeller av fordon, förare, väg och trafik samt deras samverkan ska utarbetas. Dessa modeller implementeras som moduler i dator.
- ☐ Simuleringsverktyget är avsett för **deterministisk analys**, men ska ha ett öppet gränssnitt mot makroprogrammering, t. ex. för studier av statistiska parametrars inverkan och för optimering.
- ☐ Simuleringsverktyget förutsätts vara **datorbaserat**.
- ☐ Modellerna ska kunna beskrivas av **moduler**, med tekniskt lämpliga gränssnitt.
- ☐ **Fordonet** ställs i centrum, men programskalet ska även ta ansvar för samverkan med förare, väg och trafik.
- ☐ **Tidsförlopp** ska studeras. Utöver rent algebraiska (kvasi-stationära) analyser ska även differentiella (transienta, begynnelsevärdesberoende) analyser kunna genomföras. Systemens tidskonstanter förväntas spänna över intervallet 1–100 sekunder. Även diskreta händelser (tidskonstant=0) ska kunna hanteras.
- ☐ Med dynamik i **färdriktningen** avses allt som har med framdrivning/bromsning att skaffa. Styrmanövrer och vertikala svängningar avses inte att inkluderas. Problem med tidsskalan 1–100 sekunder ska kunna hanteras.
- ☐ Projektet ska i första hand utveckla ett **programskal** med väldefinierade gränssnitt mellan moduler, i syfte att underlätta komplettering med nya moduler.
- ☐ De moduler som utvecklas inom projektet ska utgöra ett **grundbibliotek**. Där ska finnas exempel på moduler för fordon, förare, körcykler, vägar och trafik. En fordonsmodul kommer att definieras av flera submoduler, till exempel motor-, transmissions- och vagnsmodul. Modulerna som projektet utvecklar kan vara enkla. Det viktigaste är att de kan kommunicera med andra moduler och att denna kommunikation har sådant format att den är relevant även för framtidens system och frågeställningar (på minst 10 års sikt).

Simuleringsverktyget kommer i första hand att medge studier av emissioner och energianvändning vid färd enligt körcykel. Det ska vara enkelt att jämföra olika situationer och framdrivningskoncept: till exempel olika körcykler, olika motorer, olika växlingsstrategier, olika transmissioner och olika former av energilagring.

Några övriga frågeställningar som kommer att kunna hanteras är:

- ☐ Bromsförlopp, till exempel strategier för motor- och/eller drivlinebromsning.
- ☐ Startförlopp med bedömning av komfort och prestanda.
- ☐ Bedömning av växlingskvalitet under växlingsförlopp.

- ☐ Studier av tekniska detaljer, till exempel hållfasthet, slitage och värmeutveckling.
- ☐ Motorspecifika förlopp, såsom uppvärmning, emissionsbildning och avgasrening.

För de flesta studier kommer användaren dock att behöva utveckla egna moduler.

Presentation, överföring och användning av resultat

Projektets mjukvara kommer att hållas inom projektgruppen under de två åren som denna projektplan omfattar samt två år därefter. Resultat av forskningskaraktär kan komma att presenteras på konferenser och i tidskrifter även under projektiden, förutsatt att projektdeltagarna godtar offentliggörande av innehållet i det som presenteras.

Efter projektiden kommer mjukvaran att utvecklas hos respektive projektdeltagare. I första hand kommer detta att innebära modulutveckling. Resultatet kommer att användas i produktutveckling, utredningar och forskning/undervisning.

Se även projektavtal, punkterna 9 *Sekretess* och 10 *Publicering*.

Projektdeltagare och deras ansvarsområden

Projektets mål är ett programskal. För att ge skalet lämpliga former måste modulernas innehåll förutses, vilket praktiskt sett kräver att en grunduppsättning av moduler utvecklas inom projektet och att framtida, mer avancerade, moduler skisseras. Arbetsinsatser av två typer behövs således. Dessa kan benämnas *utveckling av programskal* och *definition av delsystem*. Projektdeltagarna och deras ansvarsområden framgår av totalbudgeten. Dessutom förutses projektledning och inköp av specialistkompetens som separata ansvarsområden, enligt totalbudgeten.

De industriellt verksamma deltagarna samt VTI behövs framförallt för att bidra med praktisk produkt- och problemkunskap samt provdata. Industrin ställer även en del färdiga beräkningsmodeller till projektets förfogande.

Projektdeltagarna anges i följande tabell. Ordförande i projektets styrgrupp är Mart Mägi. Projektledare är Bengt Jacobson, Maskin & fordon, CTH, tel: 031 – 772 13 83, fax: 031 – 772 13 75.

deltagare	kontaktperson	telefon	fax
Maskin & fordon, CTH	Mart Mägi	031-772 13 62	031-772 13 75
Förbränningsmotorer, LTH	Rolf Egnell	046-10 45 64	046-10 47 17
Trafikteknik, LTH	Börje Thunberg	046-10 45 70	046-12 32 72
VTI	Börje Thunberg	013-20 43 07	013-20 40 82
Volvo Lastvagnar AB	Göran Axbrink	031-66 42 31	031-23 89 78
Volvo Personvagnar AB	Stephen Wallman	031-59 56 77	031-59 63 70
Saab Automobile AB	Stefan Dunert	0520-780 10	0520-780 01
Aspen Utveckling AB	Rolf Egnell	046-18 96 20	046-18 96 25

Totalbudget

Total projektbudget [kkkr totalt under projektet]		
ansvarsområde	ansvarstagare	kostnad
projektledning och programskal	Maskin & fordon, CTH	800
delsystemet motor	Förbränningsmotorer, LTH	1200
delsystemet transmission och vagn	Maskin & fordon, CTH	1200
delsystemet trafik, väg och förare	Trafikteknik, LTH	400
specialistkompetens etc.	projektets styrgrupp	400
Subtotalt		4000
Industristöd, lika delar från Volvo LV, Volvo PV och Saab Automobile		3600
Industristöd från Aspen Utvecklings AB		400
Stöd från VTI		700
Totalt		8700

Institutionernas kostnader fördelar sig jämnt över kalenderåren 1995 och 1996. Styrgruppens post om 400 kr fördelas enligt styrgruppens delbudget nedan. Det ger följande fördelningen av sökta medel per statliga budgetår:

För projektet sökta medel [kkkr]	budgetår	
	94/95	950701-961231
Maskin & fordon, CTH	500	1500
Förbränningsmotorer, LTH	300	900
Trafikteknik, LTH	100	300
Projektets styrgrupp	100+75	225
Totalt	1075	2925
Summa sökta medel	4000	

Från Svenskt fordonstekniskt forskningsprogram söks alltså totalt 4000 kkr. Mot denna summa svarar industristödet (exkl. VTI) på lika mycket. Av de sökta medlen utgör ca 2/3 löner och resten kringkostnader (datorer, resor, lokaler och administration).

Delbudgetar

Totalbudgeten är förankrad i projektets styrgrupp. För varje delbudget ansvarar respektive ansvarstagare gentemot styrgruppen. Nedan visade delbudgetar, visar ett realistiskt alternativ för hur respektive ansvarstagare kan använda sina medel för att uppfylla sina åtaganden.

Budget för Maskin & fordon, CTH [kkkr totalt]	
Lönekostnad för projektledning och skalutveckling	630
Lönekostnad för kompetens inom transmission och vagn	550
Datorkostnader	120
Resekostnader	100
Subtotalt	1400
Påslag för administration och lokaler	600
Totalt	2000

Budget för Förbränningsmotorer, LTH [kkkr totalt]	
Lönekostnad för kompetens inom förbränningsmotorer	500
Datorkostnader	60
Resekostnader	40
Kompetens inom katalytisk avgasrening (köpes troligen av kemi-institution)	340
Kompetens inom elmotorer och elektrisk energilagring (köpes troligen av elektromaskin-institution)	
Subtotalt	940
Påslag för administration och lokaler	260
Totalt	1200

Budget för Trafikteknik, LTH [kkkr totalt]	
Lönekostnad för kompetens inom väg, trafik och förare	200
Datorkostnader	60
Resekostnader	40
Subtotalt	300
Påslag för administration och lokaler	100
Totalt	400

Budget för specialistkompetens etc. [kkkr]	tid	
	höst 1994	resten
Institutionernas kostnader för löner, resor, etc. under definitionsfasen (juli-dec. 1994)	100	-
Matematik-, numerik- och datorkompetens	-	300
Mjukvaru-licenser och/eller programmeringshjälp	-	
Reserv	-	
Totalt	100	300

Budget för industristöd (eller motsvarande) [kkkr totalt]		
Volvo	20% ingenjör (varav 10% industridoktorand)	200
Lastvagnar AB	provdata och modeller	1000
Volvo	10% ingenjör	100
Personvagnar AB	provdata och modeller	1100
Saab	10% ingenjör	100
Automobile AB	provdata och modeller	1100
Aspen	20% ingenjör	200
Utvecklings AB	provdata och modeller	200
Subtotalt		4000
VTI	20% ingenjör (kompetens inom väg, trafik och förare)	200
	provdata och modeller	500
Totalt		4700

Industristödet består, med benämningar enligt appendix, väsentligen av följande två typer av stöd:

- ☐ Posterna *ingenjör* (ca 15 % av industristödet) svarar mot den kommunikation som högskolorna kommer att behöva med industrin för att styra arbetet i en praktiskt användbar riktning.
- ☐ Posterna *provdata och modeller* (ca 85 % av industristödet) värderas med den kostnad projektet skulle haft för att producera motsvarande. Endast de delar av bidragen som har direkt relevans för projektet värderas.

Provdata, och visst utbud på beräkningsmodeller, för konventionella fordon, både personbilar och lastbilar, kommer att ställas till projektets förfogande. Stor vikt kommer att läggas vid att få med delsystemens transienta karaktär, såsom motorers eftersläpning p.g.a. turboladdning, uppvärmning från kallstart, katalysators tröghet. Det gäller dock att projektet inte får tillgång till information om mycket konkurrenskänsliga produkter och resultat.

Följande grova uppdelning mellan företagen har gjorts:

- **Saab Automobile:** Mätdata som underlag för modellering av motorns transienta beteende.
- **Volvo PV:** Underlag för modellering av typiska hybriddriftkomponenter och hjälppaggregat (t.ex. elektriska energilagrare, elmotorer/-generatorer och luftkonditioneringsanläggningar).
- **Volvo LV:** Beräkningsmodeller för dieselmotorer, inkluderande transient beteende m.a.p. emissioner och bränsleförbrukning.
- **Aspen:** Beräkningsmodeller för förbränningsmotorer, inkluderande stationärt beteende samt ett beräkningsprogram för fordon för kvasistationär (icke-transient) analys av hela körcykler.

Därutöver ser företagen det som självklart att tillföra projektet lättillgängliga fordonsdata såsom vagnsmassor, luft- och rullmotståndskoefficienter, utväxlingar och förluster i transmissionen, konverterdata, etc.

Tidplan

tid	verksamhet	
	programskalsnivå	modulnivå
juli-dec. 1994	Definition av projektinnehåll*	
jan.-mars 1995	Specifikation, utvärdering av alternativ och slutligen val av mjukvara*	Modellering av referensmodell (enkelt exempel), i programvaran Simulink*
april-juni 1995		Alternativ till Simulink provas för samma modell (referensmodellen)*
juli-dec. 1995	Generella regler för modulinnehåll och -gränssnitt*	Definition av praktiskt lämpligt modulinnehåll och -gränssnitt.***
jan.-juni 1996	Förfining av programskalet, t.ex. m.a.p. användarvänlighet*	Utveckling av grundbibliotek med moduler***
juli-dec. 1996	Slutdokumentation, i form av skriftliga rapporter samt en datorimplementation av simuleringsverktyget*	

Vid skalutvecklingen måste samtliga deltagares kunskaper och synpunkter sammanföras, varför projektledningen (Maskin & fordon, CTH) kommer att samla en arbetsgrupp med representanter från alla deltagare. Denna arbetsgrupp är aktiv i rutorna markerade med * i tidplanen ovan.

Momenten markerade med *** i tidplanen ovan, bedöms att kunna delas upp i tre arbetsgrupper:

- ☐ **Motor** omfattande motorns mekanik, energianvändning och emissioner
Huvudintressenter: Förbränningsmotorer, Aspen
- ☐ **Transmission och vagn** omfattande växellåda/CVT, energiförbrukande kringutrustning, energilagrare och vagn med färdmotstånd
Huvudintressenter: Maskin & fordon
- ☐ **Väg, trafik och förare** omfattande beskrivning av förarbeteende och körsituation övrig trafik
Huvudintressenter: Trafikteknik och VTI

Fordonsindustrin, d.v.s. Volvo LV, Volvo PV och Saab Automobile, har ett helhetsansvar för fordonet som produkt och kommer därför att vara intressenter i alla dessa delprojekt, men främst de två första.

Delrapportering till styrgruppen sker skriftligt i informella rapporter, och muntligt vid möten i samband med avslutandet av varje ruta i tidplanen. Styrgruppen informeras också via de artiklar som forskarna publicerar för sin egna akademiska meritering.

Projektets relevans för svensk fordonsindustri

Projektets resultat kommer direkt att påverka verksamheten inom var och en av de industrier, företag, högskolor och institut som är involverade i projektet. Produktutveckling, utredningar och forskning/undervisning kommer att utvecklas positivt. Det kan dessutom underlätta kontakter mellan deltagarna. Troligtvis kommer projektets resultat också att uppmärksammas utanför projektet — både nationellt och internationellt.

Simuleringsverktyget utvecklas med ambitionen att ständigt kunna förfinas genom att grundmodulerna modifieras eller nya moduler skapas. T. ex. kan inverkan av alternativa framdrivningssystem och förarbeteenden/trafiksituationer studeras. Projektet utgör därför ett direkt stöd till utveckling och utvärdering av framtida fordon.

Projektet har ett brett stöd från svensk fordonsindustri.

Annan verksamhet av betydelse för projektet

Projektet lägger tyngdpunkten på utveckling av programskalet. De moduler som projektet ansvarar för kommer främst att vara implementationer av befintliga teorier. Utveckling av nya teorier för delsystem kan vara en aktuell sidoverksamhet. Sådana nya teorier kan knytas till projektet genom att de implementeras i form av nya moduler.

En vision av projektets resultat

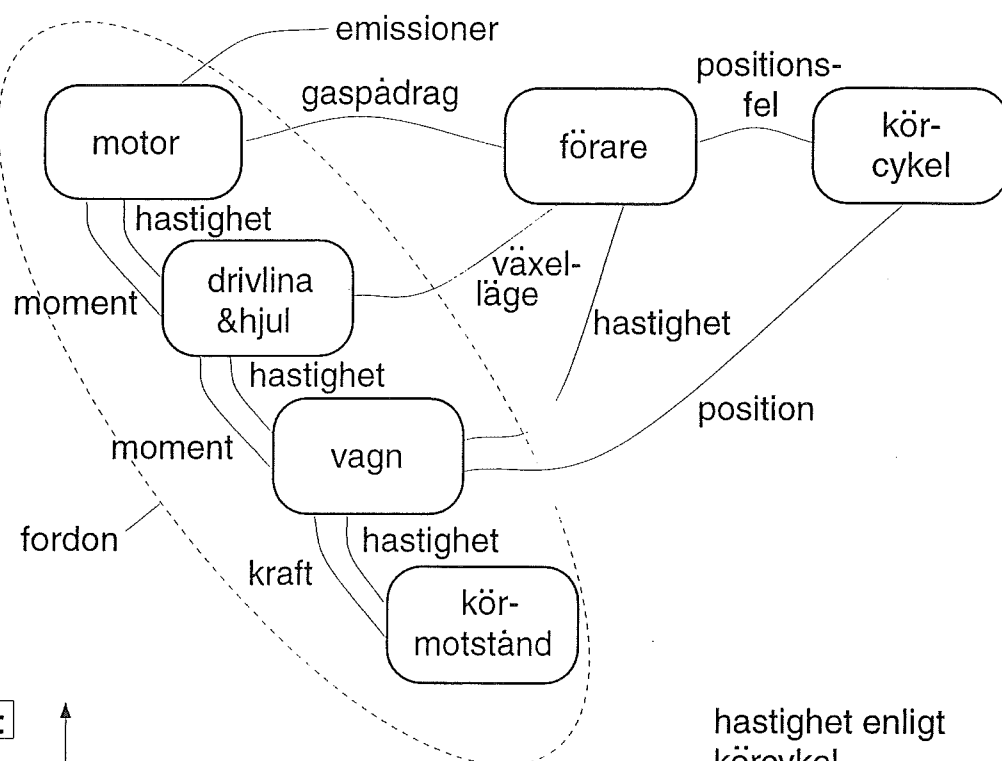
Figuren nedan visar ett exempel på moduluppbyggd modell. Moduler finns på olika hierarkiska nivåer, modulen *fordon* innehåller submodulerna *motor*, *drivlina&hjul* etc. Modulerna binds samman av gränssnittsvariabler, som ska ha en direkt fysikalisk tolkning. Till exempel har modulen *motor* gränssnittsvariablerna *emissioner*, *hastighet*, *moment* och *gaspådrag*.

Innehållet i varje modul är i princip en matematisk beskrivning av ingående komponenter. Beskrivningen ska kunna innehålla dynamiska egenskaper, det vill säga differentialekvationer som definierar problemet som ett transient problem (matematiskt sett: ett begynnelsevärdesproblem). Eftersom detta är gömt inuti modulerna, blir programmet hanterligt även för andra än simuleringsexperten.

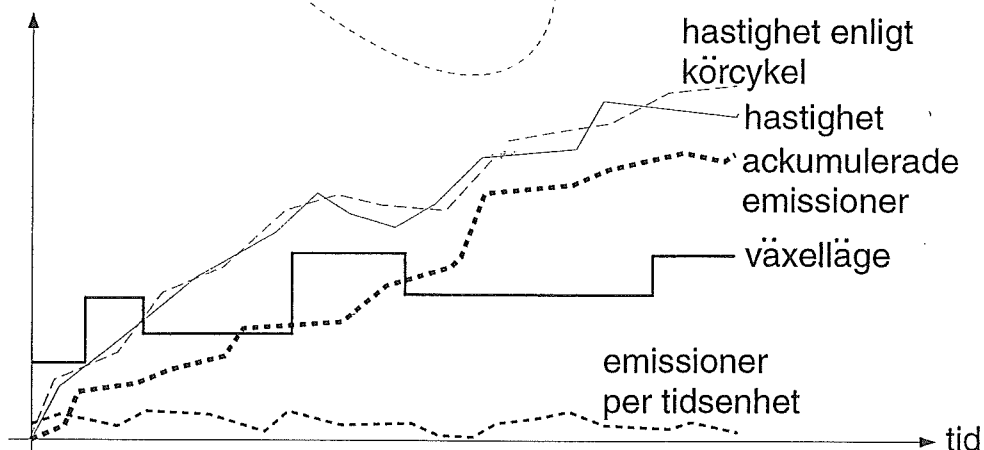
Modulerna ska vara enkelt utbytbara. Till exempel ska man kunna göra motsvarande simulering för en annan motor, som därför ska ha samma gränssnittsvariabler.

Det ska vara enkelt att definiera nya moduler. För att sörja för framtida teknik, ska även gränssnittsvariablerna kunna varieras, både till fysikalisk tolkning och antal. Som exempel kan man tänka sig att en framtida motor har ytterligare en styrsignal, utöver *gaspådrag*. Om denna styrsignal ska ges av föraren, måste naturligtvis förarmodulen förses med motsvarande gränssnittsvariabel, men övriga moduler kan hållas intakta.

MODELL:



RESULTAT:



Bengt Jacobson (projektledare)
Maskin- och fordonskonstruktion
Chalmers tekniska högskola
412 96 GÖTEBORG
Tel: 031 - 772 13 83
Fax: 031 - 772 13 75