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

Göteborg, June, 1997
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
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 an 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.
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\textsubscript{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.
Dialogue window for parameters in the engine model. (The parameter values here are valid only for this instance, i.e., this engine as a submodel in this specific vehicle model.)

Engine level. The engine is built of components (base engine, inertas, turbine, etc.) but also equations can be used (not shown here). (Changes here are propagated to all vehicle models were this engine is used.)

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\textsubscript{x} emissions (NO\textsubscript{x}S) can be compared with steady state values (NO\textsubscript{x}SS) in mass unit per second, plotted over time in seconds.

Figure 5. An Otto engine model as visible on computer screen
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 fordonsteknisk 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.
VehProp contains approximately 5 MB data and 900 files in 8 directory levels. 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
- `.dycc`: Dymola script files for an experiment (simulation) with a model. (Often it is suitable to have a script file `xxx.dycc`, 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
3.1.3 DIRECTORY COMP LIB

In principle, 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.

```plaintext
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/TL) (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.
constant ReSh=288.3
output FuelRate (kg/s)
submodel (Baseengine) Baseengine
submodel (Engcontrol) Control (fuellamp=fuellamp)
submodel (Inertial InertiaTurbo) (J=3.394e-4, wInit=wInitTurbo, wScale=1000)
submodel (Inertial InertiaCrank) (J=4.2, wInit=wInitCrank, wScale=100)
submodel (EngineBoard) Board
submodel (Compr2) Compr
submodel (Turbin2) 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*ReSh*Baseengine.mdot3**2*Turb.T03turb) (Turbine outlet pressure, [Pa])
Compr.T01=Ambient_temperature (Ambient temperature, inlet [K])
FuelRate=Baseengine.mdotq
Board.SpeedSignal=InertiaCrank.wR
end

Figure 8. File VehProp/SysLib/Vehicle/Engine/DieselDazzl.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".
General purpose software is represented by packages like SystemBuild (Reference [15]), Simulink (Reference [10]), Simmon, 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, Simmon, 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].

<table>
<thead>
<tr>
<th>METHODS FOR PROGRESS</th>
<th>STAGES OF PROGRESS</th>
<th>SOFTWARE SUPPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>strategic decision for future reuse and exchangeability of submodels</td>
<td>1. (real) system</td>
<td>physically oriented software (ADAMS, DADS, SPICE, etc.)</td>
</tr>
<tr>
<td>knowledge of important subsystem characteristics</td>
<td>2. subsystems</td>
<td>equation oriented software (Dymola, Omola, etc)</td>
</tr>
<tr>
<td>physical laws</td>
<td>3. physical (sub-)models</td>
<td></td>
</tr>
<tr>
<td>merging</td>
<td>4. mathematical (sub-)models</td>
<td>dae methods ($f(x,x)=0$) (DASSL routines, etc)</td>
</tr>
<tr>
<td>symbolic model transformation</td>
<td>5 mathematical (total) model</td>
<td>ode methods ($x=f(x)$) (MATLAB, etc)</td>
</tr>
<tr>
<td>decomposition towards original subsystems (so far it is possible)</td>
<td>6 assignment (total) model</td>
<td>assignment oriented software (Simulink, ACSL, Easy5, etc.)</td>
</tr>
<tr>
<td>integration methods for ode</td>
<td>7 assignment (sub-)models</td>
<td></td>
</tr>
<tr>
<td>interpretation of numerical result</td>
<td>8 numerical result</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9 prediction</td>
<td></td>
</tr>
</tbody>
</table>

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

**Algebraic loops** 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, **constraints** 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.

It is easy to define very **complex or even inconsistent models**. 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.

**Solution:** The software have to give good diagnostics to help the user and/or the user must have a good insight in the physical problem.

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: \( \text{speed}_1 = \text{speed}_2 = \text{speed}_3 \) and \( \text{torque}_1 + \text{torque}_2 + \text{torque}_3 = 0 \), which corresponds to the intuitive conception of the physical connection.
**Solution:** 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 change in the model library without knowing it.

**Solution:** 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.

**Solution:** 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/ToDynasi, 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 are 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, Simdon, 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.
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
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)
Examples from [Töyl 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
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/SYSIMPL (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)
- **SysTrns** (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").

<table>
<thead>
<tr>
<th>Author, Year</th>
<th>Description</th>
</tr>
</thead>
</table>


*[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.
[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.

[Stenlåås, 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
Figure 13. Planned projects and possible interaction dashed.
[12] http://www.adams.com/, information on the simulation software ADAMS from Mechanical Dynamics
[16] http://www.maplesoft.com/, information on the software Maple
<table>
<thead>
<tr>
<th>Aim or criterion</th>
<th>Dymola fulfilment</th>
</tr>
</thead>
<tbody>
<tr>
<td>The software platform should be in use for approximately 10 years</td>
<td>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.</td>
</tr>
<tr>
<td>Reasonable licence fees</td>
<td>Most licence fees, including Dymola, are reasonable compared to develop and maintain own in-house code.</td>
</tr>
<tr>
<td>Quick start for a beginner</td>
<td>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.</td>
</tr>
<tr>
<td>Support, Manuals, On-line help</td>
<td>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.</td>
</tr>
<tr>
<td>Pre- and postprocessing of simulation in general purpose numerical software, e.g., Matlab.</td>
<td>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.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Incorporation of external routines in Fortran, C, etc.</td>
<td>Dymola can incorporate C-functions in models.</td>
</tr>
<tr>
<td>Openness to other simulation software</td>
<td>Dymola generates model output on different format, such as Simstruct (for Matlab/Simulink), ACSL, Simnon.</td>
</tr>
<tr>
<td>Openness for the user</td>
<td>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.</td>
</tr>
<tr>
<td>Models on closed form for distribution without revealing model structure.</td>
<td>A compiled Dymola model can be created and distributed.</td>
</tr>
<tr>
<td>Warnings when a variable passes the range within which a model is valid.</td>
<td>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 &lt; yMax then 7 else error(&quot;message&quot;)”. Another work-around is to construct equations like “x = if y &lt; yMax then 7 else 1/0”, which makes the simulation stop by failure when y becomes &gt; yMax.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Model parameters</td>
<td>The parameter handling in Dymola is well developed. Default values can be given, parameters can be propagated between hierarchic model levels, etc.</td>
</tr>
<tr>
<td>Time and state event modelling, e.g., stick-slip phenomena for dry friction.</td>
<td>Dymola has a strong support for time and state events. It is based on a high level syntax with if- and when-statements.</td>
</tr>
<tr>
<td>Petri nets, State transition nets, or similar</td>
<td>Dymola has a library with Petri Net blocks, which is efficiently supported by the state event handler.</td>
</tr>
<tr>
<td>Model library support</td>
<td>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.</td>
</tr>
<tr>
<td>Possibility to define an arbitrary default value for an unconnected port (cut, interface, etc.)</td>
<td>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.</td>
</tr>
<tr>
<td>Data file for external tables</td>
<td>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. <strong>A better solution might be:</strong> 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.</td>
</tr>
<tr>
<td>Instantiating in base class</td>
<td>Submodels instantiated in a base class cannot be connected in an inheriting class. <strong>This is a major drawback, but will be dealt with in next Dymola version (Dymola4.0).</strong></td>
</tr>
<tr>
<td>Efficient and various integration methods available.</td>
<td>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.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Tools for Control design, System identification, Signal processing and Optimization</td>
<td>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.</td>
</tr>
<tr>
<td>Documentation support</td>
<td>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.</td>
</tr>
<tr>
<td>Debugging tools</td>
<td>The debugging features of Dymola is mainly output to a log file, dslog.txt. 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.</td>
</tr>
</tbody>
</table>
| Iteration tolerance | Cannot be controlled (except for the inconvenient way to edit the generated c-file, dsmodel.c)  
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. |
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/ToDynas**. 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/Trouble.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 Veh-Prop
- **VehProp/ReadMe/Addresses.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.
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 Dynsim AB.
   It is recommended that Dymola3.0f is used.
   (For more information on Dymola, see internet: http://www.Dynsim.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.
8) Write protection

Writing protection 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 adress of the participants in the project which developed VehProp
- VehProp\ReadMe\Trouble.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 licenses)
- Dymola -- Selected publications (shipped from Dynasim AB with Dymola licenses)
- 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.
.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.,
RECOMMENDATIONS FOR MODELS IN VehProp

In general, the following recommendations are 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 continuous 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
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)
- Temp11 (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 Temp1.cyc.

5) Wait. In the Dymola window, you will see information that the model classes are loaded and instanciated. 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, measured 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 Temp1. 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 Temp1 window. Answer yes on the question of saving to file c:\VehPropRun\Exerc1\Temp1.dym. (The saving will take some time, because Temp1.dym is a large file. Actually Temp1.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.
takes shorter time, since the parameter was changed in the Dymola window and not in the Tem1 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
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 exercise 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 "instantiations" or "instances" of model classes in the component library of VehProp. E.g., there are a compressor model class
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 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 fullfilled.
   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 operates 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
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 dures.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 Exerc4/Canned. Do File/Open TotMod.dym from the window Unnamed. You should know, that the version of Temp11 now is completely stand-alone from
3) Mark the submodel Transm in the present vehicle model. Choose Edit/ChangeSubmodelClass and write MyTransm. It is now important to note the change in the model class Temp11Vehicle. You are suggested to do File/SaveModel on Temp11Vehicle. Dymola will ask you to accept storage in file Exercs\MySym.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., MyVehicle.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 chosing 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 Temp11Vehicle. 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 Temp1.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
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 MyCompl 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 instructions 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
Table 8. Industribidrag (kSEK)

<table>
<thead>
<tr>
<th>företag</th>
<th>enligt projektplan</th>
<th>upparbetat</th>
<th>innehåll</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volvo Car Corporation</td>
<td>1 200</td>
<td>400</td>
<td>provbil</td>
</tr>
<tr>
<td></td>
<td></td>
<td>700</td>
<td>modeller av elkomponenter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>mantid</td>
</tr>
<tr>
<td>Volvo Truck Corporation</td>
<td>1 200</td>
<td>1 000</td>
<td>motormodell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>mantid</td>
</tr>
<tr>
<td>Saab Automobile</td>
<td>1 200</td>
<td>100</td>
<td>motormap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>300</td>
<td>bänkprov</td>
</tr>
<tr>
<td></td>
<td></td>
<td>150</td>
<td>CVS prov</td>
</tr>
<tr>
<td></td>
<td></td>
<td>50</td>
<td>simulering</td>
</tr>
<tr>
<td></td>
<td></td>
<td>500</td>
<td>motormap, steady state</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>mantid</td>
</tr>
<tr>
<td>Aspen Development</td>
<td>400</td>
<td>100</td>
<td>datorprogram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>100</td>
<td>transient korrigering</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
<td>mantid</td>
</tr>
<tr>
<td>summa</td>
<td>4 000</td>
<td>4 000</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 10. Från NUTEK rekviserade medel (kSEK). Sorterade efter mottagare

<table>
<thead>
<tr>
<th>Mottagare av rekviserade medel</th>
<th>budgeted</th>
<th>rekviserat</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvd, Chalmers</td>
<td>2 000</td>
<td>2 488</td>
</tr>
<tr>
<td>kraft&amp;värme, LTH</td>
<td>1 200</td>
<td>1 243</td>
</tr>
<tr>
<td>trafikteknik, LTH</td>
<td>100</td>
<td>123</td>
</tr>
<tr>
<td>VTI (formellt del av Trafik)</td>
<td>300</td>
<td>146</td>
</tr>
<tr>
<td>Styrgrupp</td>
<td>400</td>
<td>0</td>
</tr>
<tr>
<td><strong>summa</strong></td>
<td><strong>4 000</strong></td>
<td><strong>4 000</strong></td>
</tr>
</tbody>
</table>

**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 fordonssprestanda och personssäkerhet. 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.

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 haneras.

Med dynamik i **färdriktningen** avses allt som har med framdrivning/bromsning att skaffa. Styrmätningar och vertikala svängningar avses inte att inkluderas. Problemet med tidsskalan 1–100 sekunder ska kunna haneras.

Projektet ska i första hand utveckla ett **programskal** med väldefinierade gränssnitt mellan moduler, i syfte att underlätta komplettering med nya moduler.


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**.
Proektdeltagare och deras ansvarsområden

Projektets mål är ett programskal. För att ge skala lämpliga former måste modulernas innehåll förutset, 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 benämna

utveckling av programskal och definition av delsystem. Projektdeltagarna och deras ansvarsområden framgår av totalbudgeten. Dessutom förutset 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 problemkännedom samt provdata. Industrin ställer även en del färdiga beräkningsmodeller till projektets förfogande.


<table>
<thead>
<tr>
<th>deltagare</th>
<th>kontaktperson</th>
<th>telefon</th>
<th>fax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maskin &amp; fordon, CTH</td>
<td>Mart Mägi</td>
<td>031-772 13 62</td>
<td>031-772 13 75</td>
</tr>
<tr>
<td>Förbränningsmotorer, LTH</td>
<td>Rolf Egnell</td>
<td>046-10 45 64</td>
<td>046-10 47 17</td>
</tr>
<tr>
<td>Trafikteknik, LTH</td>
<td>Börje Thunberg</td>
<td>046-10 45 70</td>
<td>046-12 32 72</td>
</tr>
<tr>
<td>VTI</td>
<td>Börje Thunberg</td>
<td>013-20 43 07</td>
<td>013-20 40 82</td>
</tr>
<tr>
<td>Volvo Lastvagnar AB</td>
<td>Göran Axbrink</td>
<td>031-66 42 31</td>
<td>031-23 89 78</td>
</tr>
<tr>
<td>Volvo Personvagnar AB</td>
<td>Stephen Wallman</td>
<td>031-59 56 77</td>
<td>031-59 63 70</td>
</tr>
<tr>
<td>Saab Automobile AB</td>
<td>Stefan Dunert</td>
<td>0520-780 10</td>
<td>0520-780 01</td>
</tr>
<tr>
<td>Aspen Utveckling AB</td>
<td>Rolf Egnell</td>
<td>046-18 96 20</td>
<td>046-18 96 25</td>
</tr>
<tr>
<td>För projektet sökta medel [kkr]</td>
<td>budgetår</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>94/95</td>
<td>950701-961231</td>
<td></td>
</tr>
<tr>
<td>Maskin &amp; fordon, CTH</td>
<td>500</td>
<td>1500</td>
<td></td>
</tr>
<tr>
<td>Förbränningsmotorer, LTH</td>
<td>300</td>
<td>900</td>
<td></td>
</tr>
<tr>
<td>Trafikteknik, LTH</td>
<td>100</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>Projektets styrgrupp</td>
<td>100+75</td>
<td>225</td>
<td></td>
</tr>
<tr>
<td>Totalt</td>
<td>1075</td>
<td>2925</td>
<td></td>
</tr>
<tr>
<td>Summa sökta medel</td>
<td>4000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Historisk kostnad för kompetens inom processutveckling</th>
<th>544</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datorkostnader</td>
<td>60</td>
</tr>
<tr>
<td>Resekostnader</td>
<td>40</td>
</tr>
<tr>
<td>Kompetens inom katalytisk avgasrening (köpes troligen av kemi-institution)</td>
<td>340</td>
</tr>
<tr>
<td>Kompetens inom elmotorer och elektrisk energilagring (köpes troligen av elektromaskin-institution)</td>
<td></td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td><strong>940</strong></td>
</tr>
<tr>
<td>Påslag för administration och lokaler</td>
<td>260</td>
</tr>
<tr>
<td><strong>Totalt</strong></td>
<td><strong>1200</strong></td>
</tr>
</tbody>
</table>

**Budget för Trafikteknik, LTH [kkr totalt]**

<table>
<thead>
<tr>
<th>Lönekostnad för kompetens inom väg, trafik och förare</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datorkostnader</td>
<td>60</td>
</tr>
<tr>
<td>Resekostnader</td>
<td>40</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td><strong>300</strong></td>
</tr>
<tr>
<td>Påslag för administration och lokaler</td>
<td>100</td>
</tr>
<tr>
<td><strong>Totalt</strong></td>
<td><strong>400</strong></td>
</tr>
</tbody>
</table>

**Budget för specialistkompetens etc. [kkr]**

<table>
<thead>
<tr>
<th></th>
<th>tid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>höst 1994</td>
</tr>
<tr>
<td>Institutionernas kostnader för löner, resor, etc. under definitionsfasen (juli-dec. 1994)</td>
<td>100</td>
</tr>
<tr>
<td>Matematik-, numerik- och datorkompetens</td>
<td>-</td>
</tr>
<tr>
<td>Mjukvaru-licenser och/eller programmeringshjälp</td>
<td>-</td>
</tr>
<tr>
<td>Reserv</td>
<td>-</td>
</tr>
<tr>
<td><strong>Totalt</strong></td>
<td><strong>100</strong></td>
</tr>
<tr>
<td>VTI</td>
<td>20% ingenjör (kompetens inom väg, trafik och förare)</td>
</tr>
<tr>
<td>-----</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>prodata och modeller</td>
</tr>
<tr>
<td>Totalt</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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 transiensta beteende.
- **Volvo PV**: Underlag för modellering av typiska hybriddriftkomponenter och hjälpaggregat (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 kvasestationär (icke-transient) analys av hela körcykler.
<table>
<thead>
<tr>
<th>Period</th>
<th>Beteckning</th>
<th>Kommentar</th>
</tr>
</thead>
<tbody>
<tr>
<td>jan.-marts 1995</td>
<td>Specifikation, utvärdering av alternativ och slutligen val av mjukvara*</td>
<td>(enkelt exempel), i programvaran Simulink*</td>
</tr>
<tr>
<td>april-juni 1995</td>
<td></td>
<td>Alternativ till Simulink provas för samma modell (referensmodellen)*</td>
</tr>
<tr>
<td>Juli-dec. 1995</td>
<td>Generella regler för modulinnehåll och -gränssnitt*</td>
<td>Definition av praktiskt lämpligt modulinnehåll och -gränssnitt***</td>
</tr>
<tr>
<td>Jan.-juni 1996</td>
<td>Förfinning av programskala, t.ex. m.a.p. användarvänlighet*</td>
<td>Utveckling av grundbibliotek med moduler***</td>
</tr>
<tr>
<td>Juli-dec. 1996</td>
<td>Slutförande dokumentation, i form av skriftliga rapporter samt en datorimplementation av simuleringsverktyget*</td>
<td></td>
</tr>
</tbody>
</table>

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, energilagrar och vagn med färdmotstånd  
  Huvudintressenter: Maskin & fordon

- **Väg, trafik och förare** omfattande beskrivning av förarbetande och körsituation övrig trafik  
  Huvudintressenter: Trafikteknik och VTI
Produktutveckling, utveckling av nya tekniker och metoder, är alltid positivt. Det kan dessutom underlätta kontakter mellan deltagarna. Troligtvis kommer projektets resultat också att uppmärksammas utanför projektet — både nationellt och internationellt.


Projektet har ett brett stöd från svensk fordonssindustri.

**Annan verksamhet av betydelse för projektet**

Projektet lägger tyngdpunkten på utveckling av programskal. 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 moduluppbryggd modell. Moduler finns på olika hierarkiska nivåer, modulen *fordon* innehåller submodulerna *motor*, *drivlina* & *hjul* etc. Modulerna binds samman av gränssnittsvariblär, som ska ha en direkt fysikalisk tolkning. Till exempel har modulen *motor* gränssnittsvariblärna *emissioner*, *hastighet*, *moment* och *gaspådrag*.
